USER-BASED RECOMMENDATION ALGORITHM ON HADOOP CLUSTER

by

Sudha Varanasi

A thesis submitted to the Graduate Faculty of
Auburn University
in partial fulfillment of the
requirements for the Degree of
Master of Science

Auburn, Alabama
May 10, 2015

Keywords: Hadoop, HDFS, MapReduce, Recommendation Systems, Collaborative Filtering, User-based Similarity, Distributed Computing, Cloud Computing, Cluster

Approved by

Xiao Qin, Chair, Associate Professor of Computer Science and Software Engineering
James Cross, Professor of Computer Science and Software Engineering
Sanjeev Baskiyar, Associate Professor of Computer Science and Software Engineering

Abstract

Recommender systems apply knowledge discovery techniques to the problem of making personalized product recommendations using customers usage pattern. Systems like the k-nearest neighbors and neighborhood-based collaborative filtering are achieving widespread success in E-commerce nowadays. The tremendous growth of customers and products in recent years poses some key challenges for recommender systems. They are producing high quality recommendations and performing many recommendations per second, for millions of customers and products. New recommender systems technologies are needed to quickly produce high quality recommendations, even for very large-scale problems. A sequential implementation of deriving recommendations for a large user base has severe performance issues. We address performance issues by implementing the algorithm using Hadoop Map-Reduce framework combined with similarity based collaborative filtering techniques. Map-Reduce is a programming model and an associated implementation for processing and generating large data sets. Users specify a map function that processes a key/value pair to generate a set of intermediate key/value pairs, and a reduce function that merges all intermediate values associated with the same intermediate key. Similarity among user pairs is computed for all the users using map/reduce programming approach. Using the similarity ranking of each item is computed. The ratings of the users with highest degree of similarity produce higher ranking and are given more priority for recommendation. Map jobs will gather the compute recommendation ratings for all products on multiple nodes and reduce jobs combine the results from all the nodes to form the effective recommendation list. In this study, we will focus on user-based collaborative filtering methods, which are well known techniques used in recommender systems using Hadoop Map-Reduce. Neighbourhood-based collaborative filtering methods are user-based and item based, meaning user preferences are inferred solely from

what items they and other users in the dataset have interacted with. Experiments prove that the implementation of algorithm on Hadoop has higher performance with the increase in number of data nodes when compared to the results of implementation in a single node.

Acknowledgments

This thesis would not have been completed without invaluable guidance, experience sharing, constant support and encouragement from my advisor, people in our research group and family members during my study at Auburn University.

First and foremost I offer my sincerest gratitude to my advisor Dr. Xiao Qin, who has supported me throughout my thesis with his patience and knowledge whilst allowing me the room to work in my own way. I attribute the level of my Masters degree to his encouragement and effort and without him this thesis would not have been completed or written. One simply could not wish for a better or friendlier advisor.

I am also grateful to Dr. James Cross and Dr. Sanjeev Baskiyar for serving as members of my advisory committee. I would like to thank the Department of Computer Science and Software Engineering and Auburn University for providing such great resources and facilities.

A very special thanks goes out to my research group members without their support and help I would not have been able to finish my research. I doubt that I will ever be able to convey my appreciation fully, but I owe everyone of the group my eternal gratitude.

Finally, I would like to thank my family for the love, encouragement and support they provided me through my entire life to achieve my goals.

Table of Contents

List of Figures

List of Tables

Chapter 1

Introduction

Recommendations have become extremely common in recent years, and are applied in a variety of applications. The most popular ones are probably movies, music, news, books, research articles, search queries, social tags, and products in general. Most recommendation algorithms start by finding a set of customers who purchased and rated items overlap with the users purchased and rated items. The algorithm aggregates items from these similar customers, eliminates items the user has already purchased or rated, and recommends the remaining items to the user. To generate accurate and appropriate recommendations, the systems have to process large data sets. Data-intensive applications like these need to access ever-expanding data sets ranging from a few gigabytes to several terabytes or even petabytes. Google, for example, leverages the Map-Reduce model to process approximately twenty petabytes of data per day in a parallel fashion. The Map-Reduce programming framework can simplify the complexity of running parallel data processing functions across multiple computing nodes in a cluster, because scalable Map-Reduce helps programmers to distribute programs and have them executed in parallel. Map-reduce automatically handles the gathering of results across multiple machines and returns a single result or set. More importantly, the Map-Reduce platform can offer fault tolerance that is entirely transparent to programmers. Right now, Map-Reduce is a practical and attractive programming model for parallel data processing in high-performance cluster computing environments.

1.1 Recommender Systems

Recommender systems or recommendation systems (sometimes replacing "system" with a synonym such as platform or engine) are a subclass of information filtering system that

seek to predict the 'rating' or 'preference' that user would give to an item. Recommendation systems changed the way inanimate websites communicate with their users. Rather than providing a static experience in which users search for and potentially buy products, recommender systems increase interaction to provide a richer experience. Recommender systems identify recommendations autonomously for individual users based on past purchases and searches, and on other users' behaviour. Instead of providing a generic experience to every user, recommender systems personalize the experience of each user by surfacing content that is particularly relevant to their observed interests. If implemented correctly, recommender systems can be extremely effective at increasing engagement and purchasing. Today, many of the worlds most heavily trafficked websites, such as Netflix, LinkedIn, Amazon, and Twitter employ recommender systems to engage their users with relevant and personalized content.

1.2 Approaches to solve recommendation problem

Recommender systems apply the knowledge discovery techniques to the problem of making personalized recommendations. Recommender systems aim to solve this problem by taking in a users past actions, such as articles theyve read or products theyve purchased and rated, to identify potential user preferences. Recommender algorithms are software tools and techniques providing suggestions for items to be of use to a user. The suggestions relate to various decision-making processes, such as what items to buy, what music to listen to, or what online news to read. Many different algorithmic approaches have been applied to the basic problem of making accurate and efficient recommender systems. Few of them are described below.

1.2.1 Data Mining

Data Mining is an analytic process designed to explore data (usually large amounts of data - typically business or market related in search of consistent patterns and/or systematic relationships between variables, and then to validate the findings by applying the detected

patterns. The overall goal of the data mining process is to extract information from a data set and transform it into an understandable structure for further use. Data mining algorithms can also be used to solve recommendation system problems.

1.2.2 Machine Learning

Machine learning is a scientific discipline that explores the construction and study of algorithms that can learn from data. Such algorithms operate by building a model from example inputs and using that to make predictions or decisions, rather than following strictly static program instructions. Many of the algorithms used to solve recommendation systems problems come from the field of machine learning, a sub-field of artificial intelligence that produces algorithms for learning, prediction, and decision-making. Collaborative Filtering algorithms are one category of machine learning algorithms.

1.3 Collaborative Filtering

Collaborative filtering is the most successful recommender system technology to date, and is used in many of the most successful recommender systems on the Web. Collaborative filtering systems recommend products to a target customer based on the opinions of other customers. These systems employ statistical techniques to find a set of customers known as neighbours, that have a history of agreeing with the target user (i.e., they either rate different products similarly or they tend to buy similar set of products). Once a neighbourhood of users is formed, these systems use several algorithms to produce recommendations. To understand the algorithm and the recommendation process better, it is beneficial to introduce basic terms and become familiar with approaches, methods, goals, tasks, challenges and evaluation of recommender systems.

Items and users are the two principal entities employed in every recommender system. The term item refers to any product such as a song, a movie, a software package or an

activity that recommendation system is to recommend. Users are people willing to utilize recommendations when providing opinions about various items. The goal of collaborative filtering algorithms is to suggest new items or to predict the utility of certain item for a particular user based on users previous behaviour and similarity with others users'. Prediction is a numeric value expressing the predicted likeliness of an item for the active user. Recommendation is a list of items that the active users will like most. This is also known as top N recommendations. The collaborative filtering algorithms can further be classified into two main categories Model based (item-based) and Memory based (user-based). Collaborative filtering methods means user preferences are inferred solely from the items they and other users in the dataset have interacted with. Model-based approaches predict the rating of the items by analysing part of the data set, where as the Memory-based approach uses the complete data set to derive recommendations and hence it is more accurate.

1.3.1 Model based

Model based collaborative filtering provide item based recommendation by first developing a model of user ratings. Algorithms in this category take a probabilistic approach and compute the expected value of a user prediction given his/her ratings on the other items. The model building process is performed by different machine learning algorithms such as Bayesian network, clustering and rule based approaches.

1.3.2 Memory based

Memory based algorithms use the entire user-item database to generate raking or top n recommendations. These systems employ statistical techniques to find a set of users known as neighbours having similarities with the target user (they either rate the items similarly or tend to buy similar items). Once the similar users are identified ranking of items by these users is determined. After calculating ranking, the items with top ranks are computed. Since this type of collaborative filtering needs to process huge amounts of data, the algorithm is

designed to be implemented using Hadoop map-reduce framework to utilizing its ability of processing large volumes of data.

1.3.3 Why memory based approach?

User-similarity based recommender has an advantage over other approach like model based in providing recommendations. If the recommendations are solely based the active user's history, they might not be able to recommend new items or new areas that the user did not visit so far. On the other hand, if the recommendations are based on similarities among users, the user will receive a combined list of recommendations that will cover the areas that the user might have never accessed but still might be interested in them as the users similar to his behaviour rated them. As the memory based approach uses the entire interaction history, the accuracy of rankings of items is more likely to be high when compared with model-based approach as they predict the ratings based on part of data set.

1.4 Hadoop Map-Reduce Framework

Map-reduce is a programming model designed for processing large volumes of data in parallel by dividing the work into a set of independent tasks. The model does not work by sharing the data arbitrarily between the nodes. Instead, the data elements in the Map-reduce are immutable. The data is written only once and read many times. The data read from the input files in HDFS (Hadoop Distributed File System) are processed and converted to intermediate values and further processed to generate outputs. Any changes on the input files during this process are not reflected on the actual files.

As the name suggests Map-Reduce programs process the input data into two stages- Map stage and Reduce stage. In the mapping stage, the mapper takes one item at a time from the input list of data elements that are fetched from the HDFS and transforms to an intermediate output data element. The Map operations are paralleled when input file set

is first split into several pieces called File Splits or Input Splits. Every mapper would have exactly one input split; the number of mappers created is dependent on the number of input splits. Splitting the input file set helps in parallelizing the processing as the mappers do not have to synchronize and contend to read the file. Moreover, mappers do not have any identities of their own. Every mapper that receives the input split processes into a specified format.

The input split parser (or Record Reader) in the mapper parses the split and generates the key-value pairs. The key-value pairs are processed in parallel by the mappers, one at a time to generate exactly one intermediate key-value pair for every (key,value) pair. The output (key,value) pair of the mapper serves as input to the reducer. When the mapping phase is complete, the intermediate (key, value) pairs must be exchanged between machines to send all values with the same key to a single reducer. The reducer receives the intermediate data generated by the mapper as input, combines the values of all mapper outputs and generates a single output data corresponding to the input data fetched by the mapper. The reducers reduce a key value that is unique to each other, so reducers are same as mappers in the sense that they do not have to communicate with each other and also remain anonymous to each other.

1.5 Contribution

In this research, a specific type of machine learning algorithm called collaborative filtering algorithms, is researched and a new parallel approach for user-based collaborative recommendation algorithm is proposed based on user similarity. The user-based collaborative filtering algorithm is implemented using Hadoop Map-Reduce framework by writing a sequence of map-reduce jobs to derive the raking of items. The proposed algorithm uses the Jaccard similarity to find the most accurate recommendations for any type of data with increased performance and also increased load.

The entire data set is first processed to identify the similarities among users. This algorithm initially computes similarities between users from the data set and recommends predictions based on similarities. Map reduce jobs are written to process the data in parallel on the cluster. Map jobs run on each cluster simultaneously and pass on the results to the reduce jobs. The reduce jobs combine the results from map jobs and form the final result. The map reduce approach enables the algorithm to compute the recommendations in less amount of time as there will be many mappers running in parallel and finish the computation.

Experiments are conducted by varying number of nodes and amount of data used on the clusters and metrics are recorded. The results are compared with a single node variant which shows significant improvement in performance with the use of Map-Reduce jobs on a cluster of nodes.

1.6 Organization

Before going into the implementation details of the algorithm , we give a general introduction to Map-Reduce and HDFS in Chapter 2. In Chapter 3, we discuss the motivation for the Hadoop map-reduce version of user-based collaborative filtering algorithm. In Chapter 4, we present the design of the algorithm and in Chapter 5 we discuss the implementation details of map-reduce jobs to derive the recommendation for users. In addition to that we also describe the input and output formats at each map-reduce phase of execution. In Chapter 6, we provide the experiments along with the results of this algorithmic approach. We discuss the future work that can be done in this research in Chapter 7. Finally, Chapter 8 summarizes the main contributions and findings of this research.

Chapter 2

Background

2.1 Recommendation system algorithms

Most large-scale commercial and social websites recommend options, such as products or people to connect with, to users. Recommendation engines sort through massive amounts of data to identify potential user preferences. Most recommender systems take either of two basic approaches: collaborative filtering or content-based filtering. Many algorithmic approaches are available for recommendation engines.Most of the algorithms come from the field of machine learning, a sub-field of artificial intelligence that produces algorithms for learning, prediction, and decision-making.

Recommendation algorithms are a form of unsupervised learning that can find structure in a set of seemingly random (or unlabeled) data. In general, they work by identifying similarities among items or users by calculating their distance from other users or items in a feature space. Feature space indicates the user and item data. Broadly speaking, any software system which actively suggests an item to purchase, to subscribe, or to invest can be regarded as a recommender system. In this broad sense, an advertisement can also be a recommendation. We mainly consider, however, a narrower definition of personalized recommendation system that base recommendations on user specific information. There are two main approaches to personalized recommendation systems: content-based filtering and collaborative filtering.

2.1.1 Content-based filtering

Content-based filtering, also referred to as cognitive filtering, recommends items based on a comparison between the content of the items and a user profile. It makes explicit use of domain knowledge concerning users or items. The domain knowledge may correspond to user information such as age, gender, occupation, or location, or to item information such as genre, producer, or length in the case of move recommendation. The content of each item is represented as a set of descriptors or terms, typically the words that occur in a document. The user profile is represented with the same terms and built up by analysing the content of items that have been seen by the user. The ability of a learning method to adapt to changes in the users preferences also plays an important role. The learning method has to be able to evaluate the training data, as instances do not last forever but become obsolete as the users interests change. The data that these algorithms analyse will have to be updated very frequently as the training data becomes obsolete.

2.1.2 Collaborative filtering

Collaborative filtering arrives at a recommendation that's based on a model of prior user behaviour. The model can be constructed solely from a single user's behaviour or more effectively also from the behaviour of other users who have similar traits. When it takes other users' behaviour into account, collaborative filtering uses group knowledge to form a recommendation based on like users. In essence, recommendations are based on an automatic collaboration of multiple users and filtered on those who exhibit similar preferences or behaviours. Memory-based collaborative filtering techniques rely heavily on simple similarity measures (Cosine similarity, Pearson correlation, Jaccard coefficient) to match similar people or items together. If we have a huge matrix with users in one dimension and items in another, with the cells containing votes or likes, then memory-based techniques use similarity measures on two vectors (rows or columns) of such a matrix to generate a number representing similarity.

2.2 Collaborative filtering algorithm

Collaborative Filtering algorithm is a classic personalized recommendation algorithm; its widely used in many commercial recommender systems [1]. Collaborative Filtering algorithm is an algorithm based on the following three assumptions idea: People have similar preferences and interests; their preferences and interests are stable; we can predict their choice according to their past preferences. Because of the above assumptions, the collaborative filtering algorithm is based on the comparison of one users behaviour with other users behaviour, to find his nearest neighbours, and according to his neighbours interests or preferences to predict his interests or preferences.

Collaborative filtering systems are usually categorized into two subgroups: memory-based and model-based methods. Memory-based methods simply memorize the rating matrix and issue recommendations based on the relationship between the queried user and item and the rest of the rating matrix. Model-based methods fit a parameterized model to the given rating matrix and then issue recommendations based on the fitted model. The most popular memory-based Collaborative Filtering methods are neighbourhood-based methods, which predict ratings by referring to users whose ratings are similar to the queried user, or to items that are similar to the queried item. This is motivated by the assumption that if two users have similar ratings on some items they will have similar ratings on the remaining items. Or alternatively if two items have similar ratings by a portion of the users, the two items will have similar ratings by the remaining users. Specifically, user-based Collaborative Filtering methods identify users that are similar to the queried user, and compute the desired rating to be the average ratings of these similar users. Similarly, item-based Collaborative Filtering identify items that are similar to the queried item and estimate the desired rating to be the average of the ratings of these similar items. Neighbourhood methods vary considerably in how they compute the weighted average of ratings.

The first step of collaborative filtering algorithm is to obtain the users history profile, which can be represented as a ratings matrix with each entry the rate of a user given to an item [8]. A ratings matrix consists of a table where each row represents a user, each column represents a specific item, and the number at the intersection of a row and a column represents the users rating value. The absence of a rating score at this intersection indicates that user has not yet rated the item. Owing to the existence problem of sparse scoring, we use the list to replace the matrix. The next step is to calculate the similarity between the users and identify the similar users. There are many similarity measure methods. The calculation process of Collaborative Filtering algorithm would consume intensive computing time and computer resources. When the data set is very large, the calculation process would continue for several hours or even longer. Therefore, we propose new method that is to implement the Collaborative Filtering algorithm on Hadoop platform to reduce the computation time of the similarity.

2.3 Apache Hadoop Framework

Apache Hadoop is an open source software project that enables the distributed processing of large data sets across clusters of commodity servers. It is designed to scale up from a single server to thousands of machines, with a very high degree of fault tolerance. Rather than relying on high-end hardware, the resiliency of these clusters comes from the softwares ability to detect and handle failures at the application layer. Hadoop enables a computing solution that is scalable, cost effective, and flexible and fault tolerant.

2.3.1 Hadoop Top Architecture

Hadoop is implemented using relatively simple model of Master Slave design pattern. There are two masters in the architecture, which are responsible for the controlling the slaves across the cluster. The first master is the NameNode, which is dedicated to manage the HDFS and control the slaves that store the data. Second master is JobTracker, which

manages parallel processing of HDFS data in slave nodes using the MapReduce programming model. The rest of the cluster is made up of slave nodes, which runs both DataNode and TaskTracker daemons. DataNodes obey the commands from its master NameNode and store parts of HDFS data decoupled from the meta-data in the NameNode. TaskTrackers on the other hand obeys the commands from the JobTracker and does all the computing work assigned by the JobTracker. Finally, Client machines are neither Master nor a Slave. The role of the Client machine is to give jobs to the masters to load data into HDFS, submit Map Reduce jobs describing how that data should be processed, and then retrieve or view the results of the job when it is finished.

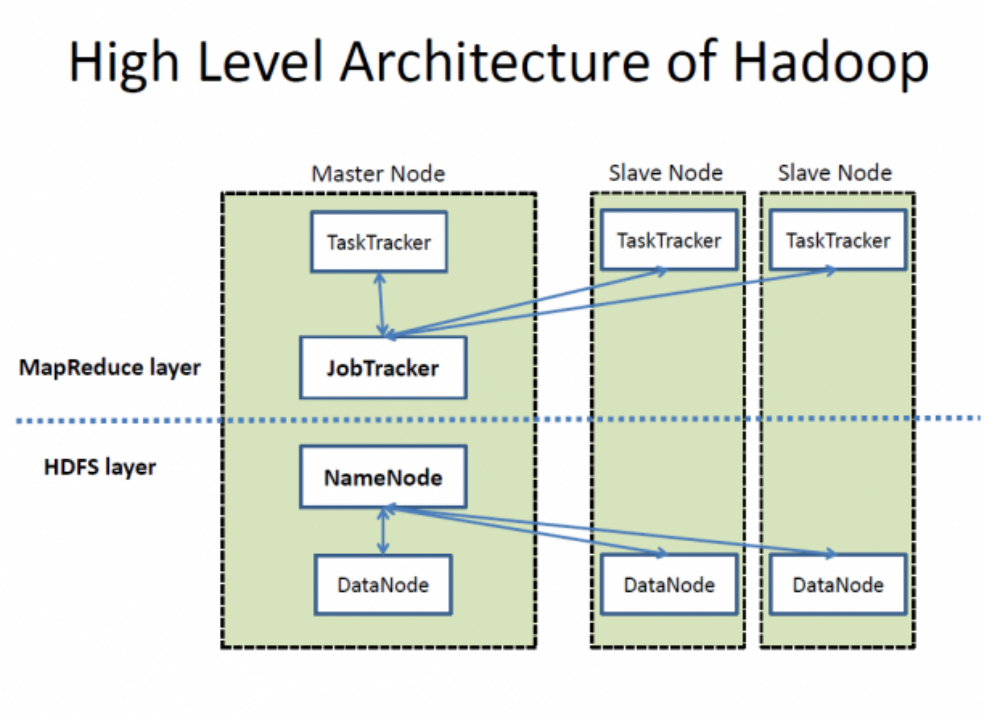

Figure 2.1: High-level Hadoop Architecture

Figure 2.1 shows the basic organization of the Hadoop cluster. The client machines communicate with the NameNode to add, move, manipulate, or delete files in HDFS. The NameNode in turn calls the DataNodes to store, delete or make replicas of data being added to HDFS. When the client machines want to process the data in the HDFS, they communicate to the JobTracker to submit a job that uses MapReduce. JobTracker divides

the jobs to map/reduce tasks and assigns it to the TaskTracker to process it. Typically, all nodes in Hadoop cluster are arranged in the air-cooled racks in a data center. The racks are linked with each other with the help of rack switches, which runs on TCP/IP.

2.4 Hadoop Distributed File System

The Hadoop Distributed File System or HDFS is a distributed file system designed to run on commodity hardware. HDFS is the primary distributed storage used by Hadoop applications on clusters. Although HDFS has many similarities with existing distributed file systems, the differences between HDFS and other systems are significant. For example, HDFS is highly fault-tolerant and is designed to deploy on cost-effective clusters. HDFS - offering high throughput access to application data - is suitable for applications that have large data sets. HDFS relaxes several POSIX requirements to enable streaming access to file system data. HDFS is not fully POSIX compliant, because the requirements for a POSIX file system differ from the design goals of Hadoop applications. HDFS trades fully POSIX compliance for increased data throughput, since HDFS was designed to handle very large files.

2.4.1 Architecture

HDFS uses master-slave architecture, in which a master is called NameNode and slaves are referred to as DataNodes. Figure 2.2 shows a diagram representing the architecture of HDFS. Basically, an HDFS cluster consists of a single NameNode, which manages the file system namespace and regulates access of clients to files. In addition, there are a number of DataNodes. Usually, each node in a cluster has one DataNode that manages storage of the node on which tasks are running. HDFS exposes file system namespace and allows user data to be stored in files. Internally, a file is split into one or more blocks stored in a set of DataNodes.

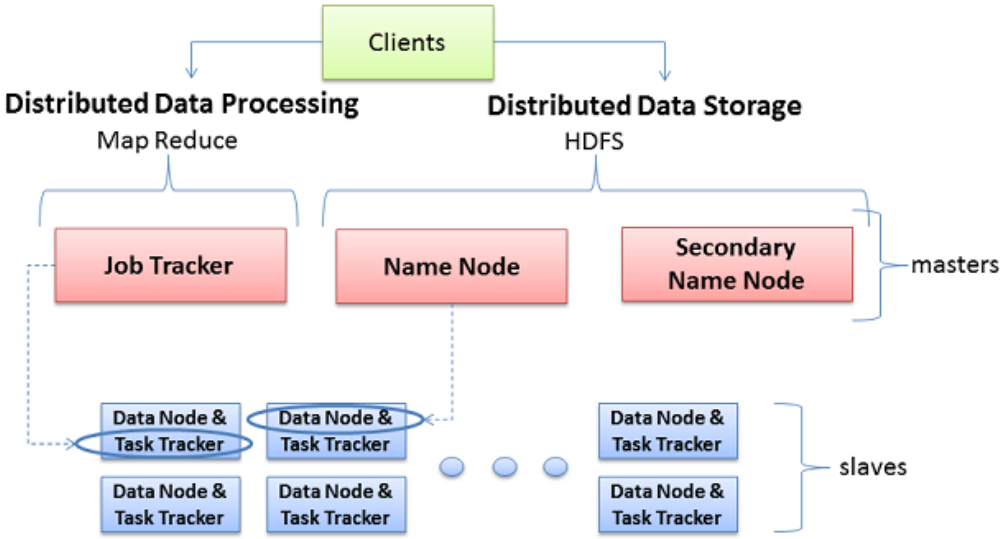

Figure 2.2: Hadoop Architecture

2.4.2 NameNode

The Namenode is the master of HDFS that maintains and manages the blocks present on the DataNodes(slave nodes). It keeps the directory tree of all files in the file system, and tracks where across the cluster the file data is kept. It does not store the data of these files itself. There is just one Namenode in Gen1 Hadoop, which is the single point of failure in the entire Hadoop HDFS cluster. The HDFS architecture is built in such a way that the user data is never stored in the Namenode.

These are the following functions of a NameNode: The NameNode maintains and executes the file system namespace. If there are any modifications in the file system namespace or in its properties, this is tracked by the NameNode

- It directs the Datanodes (Slave nodes) to execute the low-level I/O operations.
- It keeps a record of how the files in HDFS are divided into blocks, in which nodes these blocks are stored and by and large the NameNode manages cluster configuration.
- It maps a file name to a set of blocks and maps a block to the DataNodes where it is located.

- It records the metadata of all the files stored in the cluster, e.g. the location, the size of the files, permissions, hierarchy, etc.
- With the help of a transactional log, that is, the EditLog, the NameNode records each and every change that takes place to the file system metadata. For example, if a file is deleted in HDFS, the NameNode will immediately record this in the EditLog.
- The NameNode is also responsible to take care of the replication factor of all the blocks. If there is a change in the replication factor of any of the blocks, the NameNode will record this in the EditLog.
- NameNode regularly receives a Heartbeat and a Blockreport from all the DataNodes in the cluster to make sure that the datanodes are working properly. A Block Report contains a list of all blocks on a DataNode.
- In case of a datanode failure, the Namenode chooses new datanodes for new replicas, and balances disk usage and also manage the communication trafic to the datanodes.

2.4.3 DataNode

A DataNode stores data in the HDFS. A functional filesystem has more than one DataNode, with data replicated across them. On startup, a DataNode connects to the NameNode; spinning until that service comes up. It then responds to requests from the NameNode for filesystem operations. These are the following functions of a DataNode:

- Datanodes perform the low-level read and write requests from the file systems clients.
- They are also responsible for creating blocks, deleting blocks and replicating the same based on the decisions taken by the NameNode.
- They regularly send a report on all the blocks present in the cluster to the NameNode.
- Datanodes also enables pipelining of data.

- They forward data to other specified DataNodes.
- Datanodes send heartbeats to the NameNode once every 3 seconds, to report the overall health of HDFS.
- The DataNode stores each block of HDFS data in separate files in its local file system.
- When the Datanodes gets started, they scan through its local file system, creates a list of all HDFS data blocks that relate to each of these local files and send a Blockreport to the NameNode.

2.4.4 Secondary NameNode

The NameNode is the single point of failure for the Hadoop cluster, so the HDFS copies the Namespace in NameNode periodically to a persistent storage for reliability and this process is called checkpointing. Along with the NameSpace it also maintains a log of the actions that change the NameSpace, this log is called journal. The checkpoint node copies the NameSpace and journal from NameNode to applies the transactions in journal on the Namespace to create most up to date information of the namespace in NameNode. The backup node however copies the Namespace and accepts journal stream of Namespace and applies transactions on the namespace stored in its storage directory. It also stores the up-to-date information of the Namespace in memory and synchronizes itself with the NameSpace. When the NameNode fails, the HDFS picks up the Namespace from either BackupNode or CheckPointNode.

2.4.5 Replica Management Block

HDFS makes replicas of a block with a strategy to enhance both the performance and reliability. By default the replica count is 3, and it places the first block in the node of the writer, the second is placed in the same rack but difierent node and the third replica is placed in difierent rack. In the end, no DataNode contains more than one replica of a block and

no rack contains more than two replicas of same block. The nodes are chosen on the basis of proximity to the writer, to place the blocks. There are situations when the blocks might be over-replicated or under-replicated. In case of over-replication the NameNode deletes the replicas within the same rack first and from the DataNode, which has least available space. In case of under-replication, the NameNode maintains a priority queue for the blocks to replicate and the priority is high for the least replicated blocks. There are tools in HDFS to maintain the balance and integrity of the data. Balancer is a tool that balances the data placement based on the node disk utilization in the cluster. The Block Scanner is a tool used to check integrity using checksums. Distcp is a tool that is used for inter/intra cluster copying.

2.5 Map-Reduce Model

The Map-reduce model was designed for unstructured data processed by large clusters of commodity hardware; the functional style of Map-reduce automatically parallelizes and executes large jobs over a computing cluster. The Map-reduce model is capable of processing many terabytes of data on thousands of computing nodes in a cluster. Map-reduce automatically handles the messy details such as handling failures, application deployment, task duplications, and aggregation of results, thereby allowing programmers to focus on the core logic of applications. Each Map-reduce application has two major types of operations - a map operation and a reduce operation. Map-reduce allows parallel processing of the map and reduction operations in each application. Each mapping operation is independent of the others so all mappers can be performed in parallel on multiple machines. Similarly, a set of reduce operations can be performed in parallel during the reduction phase. All outputs of map operations that share the same key are presented to the same reduce operation. Map-reduce can be applied to process significantly larger datasets than commodity servers. For example, a large computing cluster can use Map-reduce to sort a petabyte of data in only a few hours. Parallelism also offers some possibility of recovering from partial failure

of computing nodes or storage units during the operation. In other words, if one mapper or reducer fails, the work can be rescheduled, assuming the input data is still available. Input data sets are, in most cases, available even in presence of storage unit failures, because each data set normally has three replicas stored in three individual storage units. A Map-Reduce program has two major phases - a map phase and a reduce phase. The map phase applies user specified logic to input data. The results, called as intermediate results, are then fed into the reducer phase so the intermediate results can be aggregated and written as a final result. The input data, intermediate result, and final result are all represented in the key/value pair format [39]. Figure 2.3 shows an executional example of the Map-Reduce model. As shown by the diagram during their respective phases multiple map and reduce jobs are executed in parallel on multiple computing nodes. Map-Reduce is also usually described in the form of the following functions summarized in Table 2.1

Input	Output
map (k1,v1)	list(k2,v2)
reduce (k2,list(v2))	list(k3,v3)

Table 2.1: Map reduce key value pairs

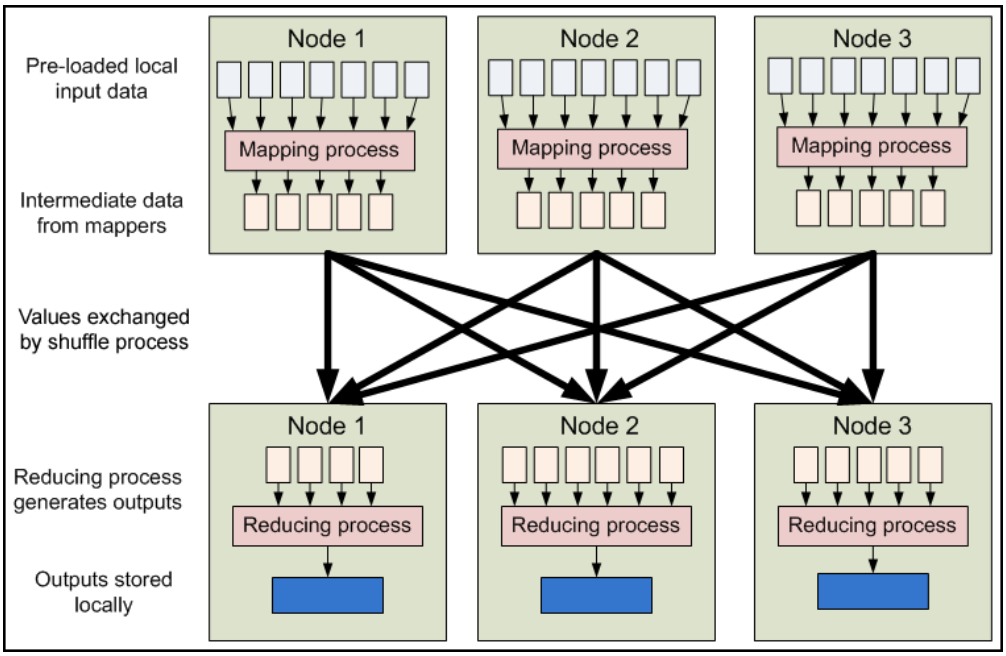

Figure 2.3: Map Reduce Model

Chapter 3

Motivation

3.1 Motivations for new approach

An increasing number of popular applications have become data-intensive in nature. In the past, the World Wide Web has been adopted as an ideal platform for developing data-intensive applications, since the communication paradigm of the Web is sufficiently open and powerful. Representative data-intensive Web applications include, but not limited to, search engines, online auctions, webmails, and online retail sales. Data-intensive applications like data mining and web indexing need to access ever-expanding data sets ranging from a few gigabytes to several terabytes or even petabytes. Google, for example, leverages the Map-Reduce model to process approximately twenty petabytes of data per day in a parallel fashion [14]. Map-Reduce is an attractive model for parallel data processing in high-performance cluster computing environments. The scalability of Map-Reduce is proven to be high, because a Map-Reduce job is partitioned into numerous small tasks running on multiple machines in a large-scale cluster.

Collaborative Filtering (CF) algorithm is a widely used personalized recommendation technique in commercial recommendation systems [7], [8], and many works have been down in this field to improve the performance. However, a big problem of CF is its scalability, i.e., when the volume of the dataset is very large, the computation cost of CF would be very high. Recently, cloud computing has been the focus to overcome the problem of large-scale computation task. Cloud computing is the provision of dynamically scalable and often virtualized resources as a service over the Internet [5]. Users need not have knowledge of, expertise in, or control over the technology infrastructure in the cloud that supports them.

Cloud computing services often provide common business applications online that are accessed from a web browser, while the software and data are stored on the servers. In order to solve scalability problem of recommender system, we implement the Collaborative Filtering algorithm on the cloud-computing platform. There are several cloud computing platforms available, for example, the Dryad [21] of Microsoft, the Dynamo [22] of amazon.com and Nettune [23] of Ask.com etc. In this paper, we choose the Hadoop platform as the base of our implementation. Because the Hadoop platform [24], [25] is an open source cloud-computing platform, it implements the Map-Reduce framework that has been successfully evaluated by Google.com. The Hadoop platform uses a distributed file system, Hadoop Distributed File System (HDFS)[26], to provide high throughput access to application data. Using the Hadoop platform, we can easily make the program execute in parallel, and the Map-Reduce framework allows the user to break a big problem into many small problems, then the small problems could be handled by the Hadoop platform, thus improving the speed of computing. The Hadoop Map-Reduce framework solves the scalability issue for systems dealing with large data sets. The ability of Hadoop framework to process huge data very fast, motivated us to utilize its capability to generate recommendations by converting the sequential approach to parallel that processes the huge data. The algorithm is divided into multiple parts to identify the components that can take the advantage of parallelization. Motivation to select the user-based collaborative filtering approach to solve the recommendation system problem is that the user-based similarities especially neighborhood algorithm derive the most accurate predictions to the user based on the tastes of similar users.

3.2 Problem Statement

Let A be U x I matrix holding all known interactions between a set of users U and a set of items I. A user u is represented by his item interaction history a-ui the u-th row of A. The top-N recommendations for this user correspond to the first N items selected from a

ranking r u of all items according to how strongly they would be preferred by the user. This ranking is inferred from patterns found in A.

3.3 Contributions

In this research, we study the implementation details of collaborative filtering algorithm on cloud computing platform. The work we have done is summarized as follows. Firstly, we designed a user based collaborative filtering algorithm for the Map-Reduce program framework, and implement the algorithm on the Hadoop platform. Secondly, we tested our implementation under several configurations. Multiple Map-Reduce jobs are written to compute similarities and derive recommendations and these jobs are run on Hadoop cluster. The user-based Collaborative Filtering algorithm that is based on users preferences is implemented using parallel programming environment of Hadoop map-reduce framework. The algorithm processes the user and item data to compute similarities and then generate recommendations for each user based on similarities computed. The idea for map reduce algorithm is based on basic map reduce paradigm i.e, to split the problem into smaller parts and compute in parallel. The data is partitioned in a way to support the parallel similarity computation. Thus taking the advantage of map-reduce the similarity computation which is the key for resource consumption is parallelized.

Chapter 4

Design

In this chapter, we discuss about user-based collaborative filtering algorithm and its map-reduce version. This section discusses the step-by-step development of our algorithmic framework. We start with showing how to conduct distributed user similarities calculation for our simple model that uses binary data. After that we generalize the approach to non-binary data to compute the rankings. Finally, we discuss how to merge the similarity matrix with user-item interaction history and generate batch recommendation. Then we can select the top ranked items to recommend. To achieve linear scalability with a growing number of users we can add more nodes to process the data.

4.1 Approach to solve problem

In order to get a clearer picture of the neighbourhood approach, it is useful to express the algorithm in terms of linear algebraic operations. Neighbourhood-based methods find and rank items that have been preferred by other users who share parts of the interaction history au. Let A be a binary matrix with Aui = 1 if a user u has interacted with an item i and Aui = 0 otherwise. In the generalization, the Aui value is the rating given by the user instead of 1.

For pairwise comparison between users, a dot product of rows of A gives the number of items that the corresponding users have in common. Similarly, a dot product of columns of A gives the number of users who have interacted with both items corresponding to the columns. When computing recommendations for a particular user with User-Based Collaborative Filtering [1], first a search for other users with similar taste is conducted. This

translates to computing the dot product of the matrix A by the users interaction history au, which results in a similarity ranking of all users with u. Secondly, the active users preference for an item is estimated by computing the weighted sum of all other users preferences for this item and the corresponding ranking. In our simple approach this translates to multiplying the rating of all users with similarity matrix computed in previous step to generate ranking.

ru gives the ranking of all items for user u and top N items with highest ranking can be selected and recommended to the user.

Notation hints: au denotes the u-th row of the interaction matrix A, ai denotes the i-th column of A, U denotes the number of users which is equal to the number of rows in A. for-each i v denotes iteration over the indexes of non-zero entries of a vector v, for-each (i, k) v denotes iteration over the indexes and the corresponding non-zero values of a vector v.

4.2 Sequential Approach

The standard sequential approach [19] for computing the similarity of users S. S (u, v) is defined as

$$Sim_{u,v} = dot(u,v)/ItemsRated_u + ItemsRated_v - dot(u,v) \tag{4.1}$$

$$dot(u,v) = vector(u).vector(v) \tag{4.2}$$

ItemsRated(u) = computes the count of items rated by users u

To get the similarity of users we need to compute the dot product of each row (user vector) of A with each column (another user vector) of A. The algorithm below shows the computation of the similarity among the users by first computing the ItemsRated by each user and then computing the dot product of users.

for-each item j rated by user u

$IR_u = IR_u + 1$

Figure 4.1: Computes items rated by the user

The algorithm shown in 4.1 computes the count of items rated by the user. This is a component used to compute similarity of a user-pair.

for-each user u do

for-each item i rated by user u

for-each user v who also rated item i do

$D_{uv} = D_{uv} + 1$

$S_{uv} = D_{uv} / IR_u + IR_v - D_{uv}$

Figure 4.2: Compute similarity between user-pair

The sequential algorithm takes more time with the increasing number of users and items. The components calculated in the previous algorithm are used here to compute the similarity. If we wish to distribute the computation across several machines on a shared-nothing cluster, this approach becomes infeasible, as it requires random access to both users and items in its inner loops. Its random access pattern cannot be realized efficiently when we have to work on partitioned data. In order to improve the runtime speed-up proportional to the number of machines in the cluster the algorithm to should be modified to a parallel version.

The standard sequential algorithm cannot be used for parallel version as the computation of the dot product of the users requires random access to the rows and columns of the user-item matrix A. The algorithm to compute the dot product is the most resource consuming part of similarity computation and has to be addressed to improve the performance.

One solution to this problem is to pre-process the data in a way that the computation can take place either by using only row or column data of the matrix. Then the data is

processed by row-wise across multiple machines and can achieve parallelism of the similarity computation. After applying the pre-processing technique we design a use map reduce version to compute the similarity and derive recommendations using the similarity matrix of the users.

4.3 Algorithm design using Map reduce

In order to scale out the similarity computation from sequential algorithm, it needs to be phrased as a parallel algorithm, to make its runtime speed-up proportional to the number of machines in the cluster. This is not possible with the standard sequential approach, as it requires random access to the rows and columns of A in the inner loops of algorithm, which cannot be efficiently realized in a distributed, shared- nothing environment where the algorithm has to work on partitioned data. We need to find a way of executing this multiplication that is better suited to the Map-Reduce paradigm and has an access pattern that is compatible to partitioned data. The solution is to split the computation into different parts.

Multiple map-reduce jobs can run mappers in parallel to compute different components of the equation and reduce jobs finally compute the similarity and ratings. Each row of the matrix represents each users information. When we transpose the matrix, each row of the user contains all the user ratings for a specific item. Because the u-th column of AT is identical to the u-th row of A, we can compute the common ratings of one user with another user by just processing each row of matrix for each item. The data is pre-processed such that all users ratings of one item is available on one node so that the computation happens with all users. Data is partitioned by hadoop row wise, each row creates a mapper, the computation for each item is parallelized and all the results are combined in reducer to get the overall rating summary of all items for each user pair. We discuss more details of the

algorithm like data pre-processing, data partitioning, map reduce jobs, input and output formats of the jobs in the further sections.

4.3.1 Data Preprocessing

To parallelize the similarity calculation, the computation needs to be carried out in small parts on multiple machines. The key is to pre-process data such that all the data needed to complete on small task need to be present in one node. In our problem, we need to compute the user-similarity for each pair of users. So all the user data should be available on one node. All the item ratings on the same node are not need to compute the similarity. Hence, the data partition can be item based. Each node can just have the data of some items. The results of these small tasks can be clubbed together to compute similarity for a pair of users.

Item-based data partition The dataset we have is a user-item matrix, which has ratings for each item given by all the users. We need to reorganize the data having each item ratings as one of row data so that the data can be partitioned in the way that benefits the algorithm.

The data shown in figure 4.3 is reorganized to be available for processing by hadoop. The data is simply the transpose of the previous matrix. It is very easy to generate the data in this format from the available format as shown in figure 4.3.

The data set above is item based and each row represents all the data for all users. Now this data can be partitioned so that the parallel computation of similarity can take place.

The data set is divided into two partitions partion1 shown in figure 4.5 and partion2 as shown in figure 4.8, each of which contains a set of items and their ratings.

User-Based Partition For deriving the recommendations from the similarity computation done in the previous stage. The results have to be merged with input data. Further partitioning and preprocessing of the input data at this stage is useful to run the jobs faster.

```
1000:1-0.0;2-0.0;3-0.0;4-0.0;5-0.0;6-0.0;7-0.0;8-0.0;9-0.0;10-0.0;11-0.0;12-0.0;13-0.0;14-0.0;15-0.0;16-0.0;17-0.0;
100:1-0.0;2-0.0;3-0.0;4-0.0;5-0.0;6-0.0;7-0.0;8-0.0;9-0.0;10-0.0;11-0.0;12-0.0;13-0.0;14-0.0;15-0.0;16-0.0;17-0.0;1
101:1-0.0;2-0.0;3-0.0;4-0.0;5-0.0;6-0.0;7-0.0;8-0.0;9-0.0;10-0.0;11-0.0;12-0.0;13-0.0;14-0.0;15-0.0;16-0.0;17-0.0;1
102:1-0.0;2-0.0;3-0.0;4-0.0;5-0.0;6-0.0;7-0.0;8-0.0;9-0.0;10-0.0;11-0.0;12-0.0;13-0.0;14-0.0;15-0.0;16-0.0;17-0.0;1
103:1-0.0;2-0.0;3-0.0;4-0.0;5-0.0;6-0.0;7-0.0;8-0.0;9-0.0;10-0.0;11-0.0;12-0.0;13-0.0;14-0.0;15-0.0;16-0.0;17-0.0;1
104:1-0.0;2-0.0;3-0.0;4-0.0;5-2.0;6-0.0;7-0.0;8-0.0;9-0.0;10-0.0;11-0.0;12-4.0;13-0.0;14-0.0;15-0.0;16-0.0;17-0.0;1
105:1-0.0;2-0.0;3-0.0;4-0.0;5-0.0;6-0.0;7-2.0;8-0.0;9-0.0;10-0.0;11-0.0;12-0.0;13-0.0;14-0.0;15-0.0;16-0.0;17-0.0;1
106:1-0.0;2-0.0;3-0.0;4-0.0;5-0.0;6-0.0;7-0.0;8-0.0;9-0.0;10-0.0;11-0.0;12-0.0;13-0.0;14-0.0;15-0.0;16-0.0;17-0.0;1
107:1-0.0;2-0.0;3-0.0;4-0.0;5-0.0;6-0.0;7-0.0;8-0.0;9-0.0;10-0.0;11-0.0;12-0.0;13-0.0;14-0.0;15-0.0;16-0.0;17-0.0;1
10:1-0.0;2-0.0;3-0.0;4-4.0;5-0.0;6-0.0;7-0.0;8-4.0;9-0.0;10-0.0;11-2.5;12-0.0;13-3.0;14-0.0;15-2.0;16-0.0;17-0.0;18
110:1-0.0;2-4.0;3-0.0;4-0.0;5-4.0;6-0.0;7-0.0;8-5.0;9-0.0;10-0.0;11-4.0;12-4.0;13-5.0;14-0.0;15-0.0;16-0.0;17-4.0;1
111:1-0.0;2-0.0;3-0.0;4-0.0;5-0.0;6-0.0;7-0.0;8-0.0;9-0.0;10-0.0;11-0.0;12-0.0;13-0.0;14-0.0;15-0.0;16-0.0;17-0.0;1
112:1-3.5;2-0.0;3-0.0;4-0.0;5-0.0;6-0.0;7-0.0;8-0.0;9-0.0;10-0.0;11-0.0;12-0.0;13-0.0;14-0.0;15-0.0;16-0.0;17-0.0;1
113:1-0.0;2-0.0;3-0.0;4-0.0;5-0.0;6-0.0;7-0.0;8-0.0;9-0.0;10-0.0;11-0.0;12-0.0;13-0.0;14-0.0;15-0.0;16-0.0;17-0.0;1
114:1-0.0;2-0.0;3-0.0;4-0.0;5-0.0;6-0.0;7-0.0;8-0.0;9-0.0;10-0.0;11-0.0;12-0.0;13-0.0;14-0.0;15-0.0;16-0.0;17-0.0;1
116:1-0.0;2-0.0;3-0.0;4-0.0;5-0.0;6-0.0;7-0.0;8-0.0;9-0.0;10-0.0;11-0.0;12-0.0;13-0.0;14-0.0;15-0.0;16-0.0;17-0.0;1
117:1-0.0;2-0.0;3-0.0;4-0.0;5-0.0;6-0.0;7-0.0;8-0.0;9-0.0;10-0.0;11-0.0;12-0.0;13-0.0;14-0.0;15-0.0;16-0.0;17-0.0;1
118:1-0.0;2-0.0;3-0.0;4-0.0;5-0.0;6-0.0;7-0.0;8-0.0;9-0.0;10-0.0;11-0.0;12-0.0;13-0.0;14-0.0;15-0.0;16-0.0;17-0.0;1
119:1-0.0;2-0.0;3-0.0;4-0.0;5-0.0;6-0.0;7-0.0;8-0.0;9-0.0;10-0.0;11-0.0;12-0.0;13-0.0;14-0.0;15-0.0;16-0.0;17-0.0;1
11:1-0.0;2-0.0;3-0.0;4-0.0;5-5.0;6-0.0;7-4.0;8-0.0;9-0.0;10-4.0;11-0.0;12-0.0;13-4.0;14-0.0;15-3.0;16-0.0;17-0.0;18
121:1-0.0;2-0.0;3-0.0;4-0.0;5-0.0;6-0.0;7-0.0;8-0.0;9-0.0;10-0.0;11-0.0;12-0.0;13-0.0;14-0.0;15-0.0;16-0.0;17-0.0;1
122:1-0.0;2-0.0;3-0.0;4-0.0;5-0.0;6-0.0;7-2.0;8-0.0;9-0.0;10-0.0;11-0.0;12-0.0;13-0.0;14-0.0;15-0.0;16-4.0;17-0.0;1
123:1-0.0;2-0.0;3-0.0;4-0.0;5-0.0;6-0.0;7-0.0;8-0.0;9-0.0;10-0.0;11-0.0;12-0.0;13-0.0;14-0.0;15-0.0;16-0.0;17-0.0;1
124:1-0.0;2-0.0;3-0.0;4-0.0;5-0.0;6-0.0;7-0.0;8-0.0;9-0.0;10-0.0;11-0.0;12-0.0;13-0.0;14-0.0;15-0.0;16-0.0;17-0.0;1
125:1-0.0;2-0.0;3-0.0;4-0.0;5-0.0;6-0.0;7-0.0;8-0.0;9-0.0;10-0.0;11-0.0;12-0.0;13-0.0;14-0.0;15-0.0;16-0.0;17-0.0;1
126:1-0.0;2-0.0;3-0.0;4-0.0;5-0.0;6-0.0;7-0.0;8-0.0;9-0.0;10-0.0;11-0.0;12-0.0;13-0.0;14-0.0;15-0.0;16-0.0;17-0.0;1
129:1-0.0;2-0.0;3-0.0;4-0.0;5-0.0;6-0.0;7-0.0;8-0.0;9-0.0;10-0.0;11-0.0;12-0.0;13-0.0;14-0.0;15-0.0;16-0.0;17-0.0;1
12:1-0.0;2-0.0;3-0.0;4-0.0;5-0.0;6-0.0;7-0.0;8-0.0;9-0.0;10-0.0;11-0.0;12-0.0;13-0.0;14-0.0;15-0.0;16-0.0;17-0.0;18
130:1-0.0;2-0.0;3-0.0;4-0.0;5-0.0;6-0.0;7-0.0;8-0.0;9-0.0;10-0.0;11-0.0;12-0.0;13-0.0;14-0.0;15-0.0;16-0.0;17-0.0;1
```

Figure 4.3: Input Data Set

Reorganized Data set (before parition)

Items\Users	User1	User2	User3	User4	User5	
Item1	R11	R21	R31	R41	R51	
Item2	R12	R22	R32	R42	R52	
Item3	R13	R23	R33	R43	R53	
Item4	R14	R24	R34	R44	R54	

Figure 4.4: Reorganized Data Set

Parition1

Items\Users	User1	User2	User3	User4	User5	
Item1	R11	R21	R31	R41	R51	
Item2	R12	R22	R32	R42	R52	
...						
...						
...						

Figure 4.5: Partitioned Data Set Part1

Partition2

Items\Users	User1	User2	User3	User4	User5	
Item3	R13	R23	R33	R43	R53	
Item4	R14	R24	R34	R44	R54	
...						
...						
.....						

Figure 4.6: Partitioned Data Set Part2

Earlier we partitioned the data item-based, now we can further partition the data user-based so that each row of the data contains ratings of a set of users. Now map-reduce jobs will merge the similarity results with the raw data partitioned using both user-based and item-based strategy. This enables to leverage the map-reduce programming advantage at more higher level.

Items\Users	User1	User2
Item3	R13	R23
Item4	R14	R24
...		
...		
.....		

Items\Users	User3	User4	User5
Item3	R33	R43	R53
Item4	R34	R44	R54
...			
...			
.....			

User based data partition

Figure 4.7: Item and User-Based partition

4.3.2 Similarity Computation

After the data is partitioned, the similarity computation should be modified such that it can be divided into sub problems, which can compute part of the result on each node. The similarity computation is divided in to two parts, the first part is to compute the dot product of each user pair per item and store the results. Next, compute the sum of all items dot product from all nodes to get the complete similarity of each pair of users.

Algorithm: to compute similarity

for-each user u, item i do

for-each user v who also rated item i do

$D_{uv} = D_{uv} + 1$

Figure 4.8: Algorithm to Compute Similarity in parallel

All the nodes that have item data should call this algorithm and results for each item are saved. Then the results are passed on to be summed up, to compute the final similarity of the user pair. The same algorithm can be used to compute the ItemsRated of each user vector by summing up the value for each item. If the user rates the item, the ItemsRated value for that user is incremented by 1.

Thus, the same algorithm can be used to compute items rated count and dot product for pair of users. Then these results are used to compute the similarity for each pair of users. This task is also handled in parallel as the data for each pair can reside on any data node and each data node will compute the similarities of the user-pair whose dot product and items rated values are present on the node.

4.3.3 Deriving Recommendations

The next part of the problem is to derive recommendations by using the similarity matrix computed in the previous phase. We can generate batch recommendations to all the

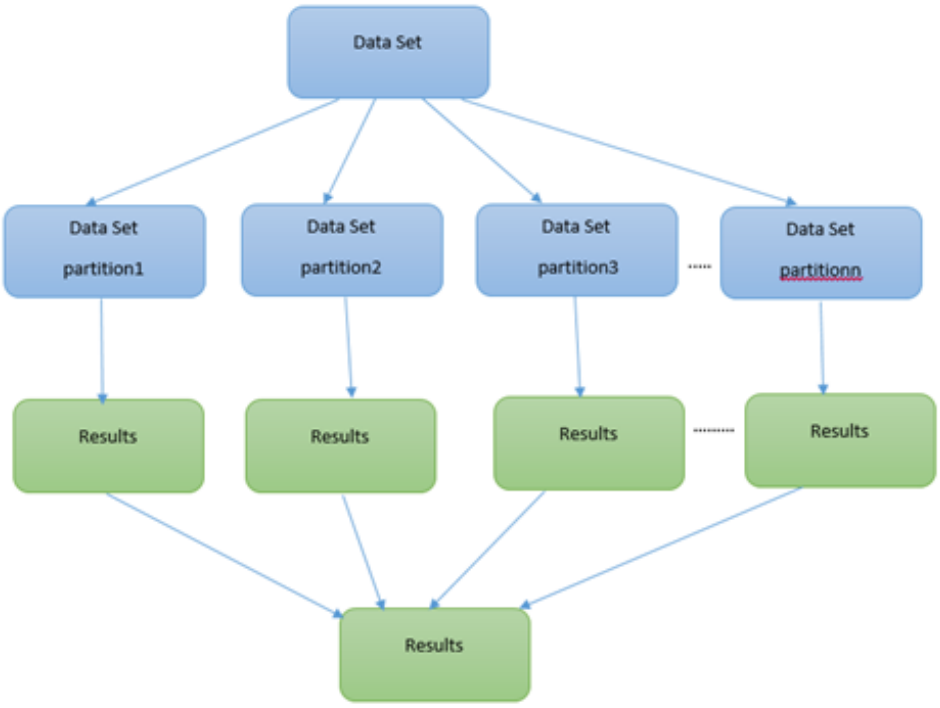

Figure 4.9: Parallel Processing of data by multiple nodes

users or a set of active users. To generate recommendations, the items ranking should be computed. To generate the item ranking for a user 'u'. The item ranking for each user pair can be computed as follows. For user pair (u, v) is ranking is computed by multiplying the similarity value and item rating given by user 'v'. This gives the ranking for user 'u' for all items that are rated by 'v' w.r.t the similarity behavior of user 'v'. This process should be repeated for all user pairs and all the individual user-item ratings are computed. Then all the user-item ratings are summed up to get the overall ranking of the item. From all the items, items with top N ranking are selected and provided as the recommendation to the user.

The figure 4.10 demonstrates how the recommendations can be derived using the similarity computation.

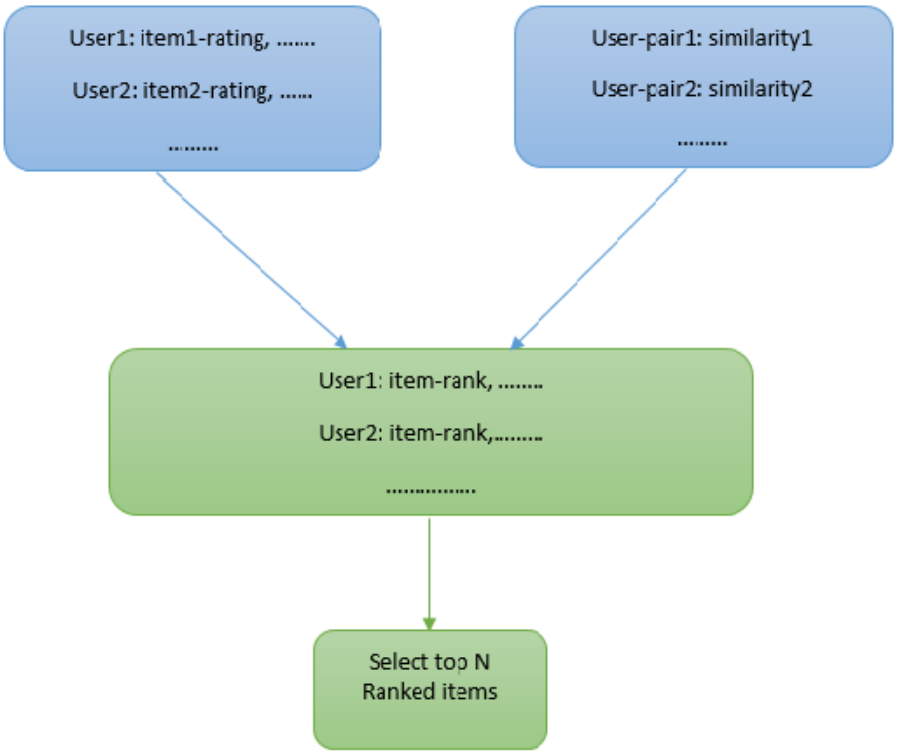

Figure 4.10: Derive Recommendation from raw data and similarity computation results

Chapter 5

Implementation

5.1 Map Reduce Jobs

As discussed earlier, map reduce jobs are tasks the run in parallel on Hadoop cluster. A main task is sub-divided into multiple smaller map-reduce jobs, where in each job executes the same task but on a different data. The huge data which is processed by Hadoop is partitioned onto several data nodes and the job tracker uses the data on the data node and executes the program. Each map task generates a (key, value) pair as intermediate output. The reduce tasks will collect all the results with same key and process them to generate the final output. Map-reduce jobs use text files as input and output instead of traditional RDBMS systems.

5.1.1 Input data for map reduce jobs

For the hadoop map reduce jobs to process the data, the data should be in the form of key value pairs. The input data is created as a text file by querying the database system. We can pre-process the data so that it can processed by hadoop map reduce jobs. The text file contains the data item-wise, each row contains ratings of all users for one item. Map-reduce jobs works based on key value pairs. In this thesis, we assume that the input data pre-processes in the required format.

The item-id is used as key and rest of the line (containing the user id and rating) as value. When a map reduce job executes, each row data is processed by an individual mapper. Input format: input format used for the map-reduce job is text file which contains key value pair delimited by a common character in each line. When Hadoop runs the map reduce job,

it processes each line as a different mapper and uses the data to create intermediate results as key value pairs.

```
m1:u1-1;u2-0;u3-1;u4-1;u5-1;u6-0;u7-1;u8-0;u9-1;u10-0
m2:u1-0;u2-0;u3-0;u4-0;u5-0;u6-0;u7-0;u8-1;u9-0;u10-0
m3:u1-1;u2-1;u3-1;u4-0;u5-1;u6-1;u7-1;u8-1;u9-1;u10-0
m4:u1-0;u2-1;u3-1;u4-0;u5-0;u6-0;u7-1;u8-1;u9-0;u10-1
m5:u1-0;u2-0;u3-0;u4-0;u5-0;u6-0;u7-0;u8-0;u9-0;u10-0
m6:u1-1;u2-1;u3-1;u4-1;u5-0;u6-1;u7-1;u8-1;u9-0;u10-1
m7:u1-0;u2-0;u3-0;u4-0;u5-0;u6-0;u7-0;u8-0;u9-0;u10-0
m8:u1-1;u2-0;u3-1;u4-1;u5-1;u6-1;u7-1;u8-1;u9-0;u10-1
m9:u1-0;u2-0;u3-0;u4-0;u5-0;u6-0;u7-1;u8-0;u9-0;u10-0
m10:u1-0;u2-0;u3-0;u4-0;u5-0;u6-0;u7-0;u8-1;u9-1;u10-1
```

Figure 5.1: Input data format after pre-processing of data

Figure 5.1 shows the input data format.

Before proceeding with the solution, as the recommendation problem is divided into two parts, and a set of map reduce jobs are written to finish each part of the problem solution.

5.2 Compute Similarity between user pairs

The similarity computation phase of the recommendation problem is again sub-divided into two map reduce jobs. The first map-reduce job computes the dot product, ItemsRated values of each pair of users. The second job computes similarity for each pair using the values computed by the previous map-reduce job.

Map-reduce Job1

Compute dot product, ItemsRated sum of a pair of user vectors. Map task: As mentioned earlier, each row of data is processed by a mapper, the map job creates user-user keys and values for product, sum of ratings of the user pair for each item. The map job creates two key value pairs for each mapper. One key represents, the product and other represents sum of ItemsRated of both the users. These results are for one item (the record processed by that mapper). The map task produces some intermediate results which are processes by the reduce job and the over-all results are generated.

Reduce task: The reduce job now collects all the user-user-product keys and user-user-sum keys and sum up the results to get the overall all dot product and ItemsRated for all pairs of users.

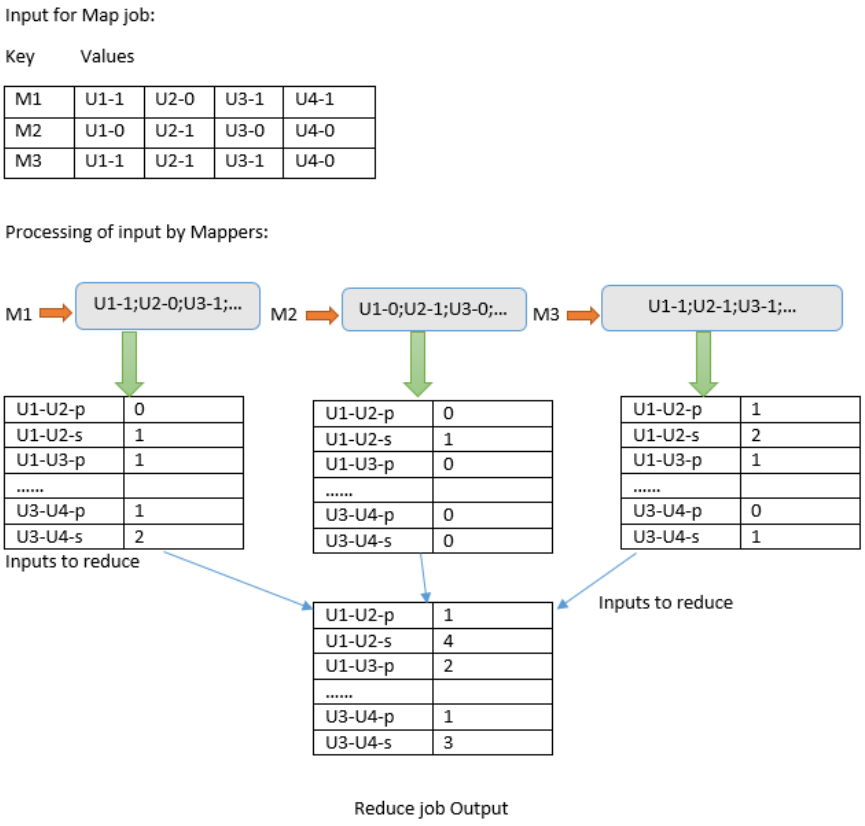

Figure 5.2: MRJob1-Compute the components for similarity calculation

The figure 5.2 shows the inputs and outputs for each mapper and reducer.

The figure 5.3 shows the final output for first map reduce job.

5.2.1 Map-reduce Job2

The next map reduce job will take the output of the previous map reduce job as input and computes the similarity of each pair of users. The map task gets user pair, product/sum as key and product/sum value respectively. The map task then splits the key as user pair

```
u1-u10,p@       2
u1-u10,s@       8
u1-u2,p@        2
u1-u2,s@        7
u1-u3,p@        4
u1-u3,s@        9
u1-u4,p@        3
u1-u4,s@        7
u1-u5,p@        3
u1-u5,s@        7
u1-u6,p@        3
u1-u6,s@        7
u1-u7,p@        4
u1-u7,s@        10
u1-u8,p@        3
u1-u8,s@        10
u1-u9,p@        2
u1-u9,s@        7
u2-u10,p@       2
u2-u10,s@       7
u2-u3,p@        3
u2-u3,s@        8
u2-u4,p@        1
u2-u4,s@        6
u2-u5,p@        1
```

Figure 5.3: Output of MR1 with sample data set

and includes the product/sum part of the key to value. Map task outputs the user pairs as key and product or ItemsRated sum values as value. Each user pair has 2 records, one with product and other sum. This makes it possible for the reduce job to collect the same user-pair and then compute the similarity of the user by using the product and sum components of the user pair.

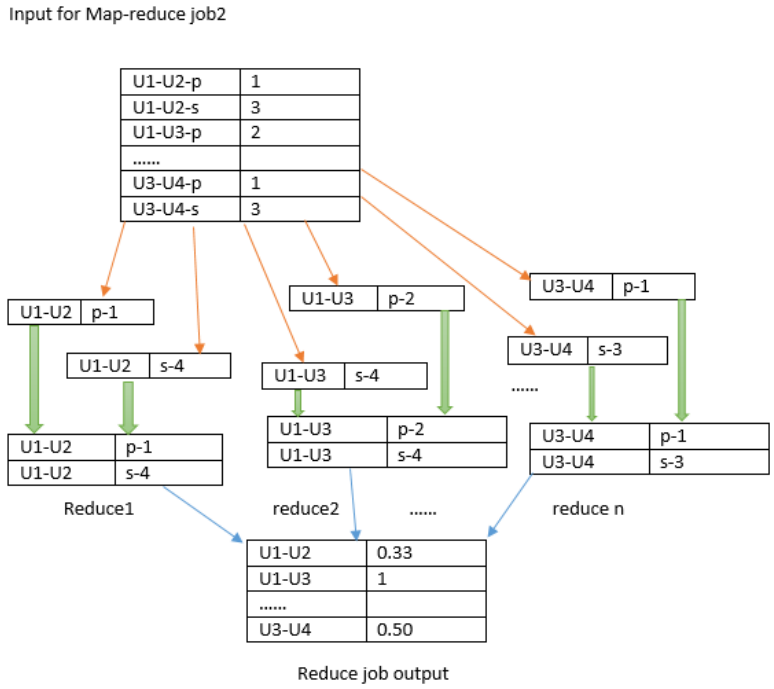

Figure 5.4: MRJob2 - compute similarity using the components

The figure 5.4 shows the inputs and outputs for each mapper and reducer for second map reduce job.

Reduce task: Combines the dot product and ItemsRated sum components of the user-pair and computes the similarity. The reduce job generates user pair and key and similarity as value output.

The figure 5.15 shows the final output for second map reduce job.

The first two map reduce jobs complete the first part of the problem, to compute similarity of each user pair. By using map-reduce job, we reduced the processing time of

```
invalidString@  -1.0
invalid -1.0
u1-u10@ 0.33
u10-u1@ 0.33
u1-u2@  0.4
u2-u1@  0.4
u1-u3@  0.8
u3-u1@  0.8
u1-u4@  0.75
u4-u1@  0.75
u1-u5@  0.75
u5-u1@  0.75
u1-u6@  0.75
u6-u1@  0.75
u1-u7@  0.67
u7-u1@  0.67
u1-u8@  0.43
u8-u1@  0.43
u1-u9@  0.4
u9-u1@  0.4
u2-u10@ 0.4
u10-u2@ 0.4
u2-u3@  0.6
u3-u2@  0.6
u2-u4@  0.2
u4-u2@  0.2
```

Figure 5.5: Output of MR2 with sample data set

the computation by a significant amount of time as the computation is mostly parallelized at item level in the first map-reduce job and user-pair level next in the second map-reduce job. This ensures that the computation is quick when compared to the sequential method of similarity computation.

5.3 Derive batch recommendation

The next major part of the problem is to derive recommendations for each user in the system based on the items already rated/ not-rated and also choose the item based on user-similarity. The item which is rated by the most similar user is given a higher ranking and this it gets highest priority in the list of non-rated items. To compute this we need to combine the user-item data with the user-similarity and then compute rankings based on the similarity. Then sort the items based on their ranks and select the top N items and provide the user with these item details.

5.3.1 Map-reduce Job3

The third map-reduce job is to combine the user-pairs and the user-item data to generate records that can be processed in parallel. The map job reads two input files, each file processes its data and generates the same key but a different value. This enables the reducer to combine the records from map-reduce and then produce the required output format. To process files multiple locations we use MultipleInputs class provided by Hadoop.

The map job takes the user item data and output of previous reduce job as inputs and process these records to gets user as key and values will be all items with ratings from first file and the other file produces user as key and value as other user-similarity value.

The outputs from these two files will have same key but different values. The reduce job then combines all the items with same key and produces a combined values as output. The reduce job creates key for each user item pair and has the other users similarity as value.

Here the user-item are formed for only the items we are interested to recommend to the user, so the key contains only the items that are not rated by the user.

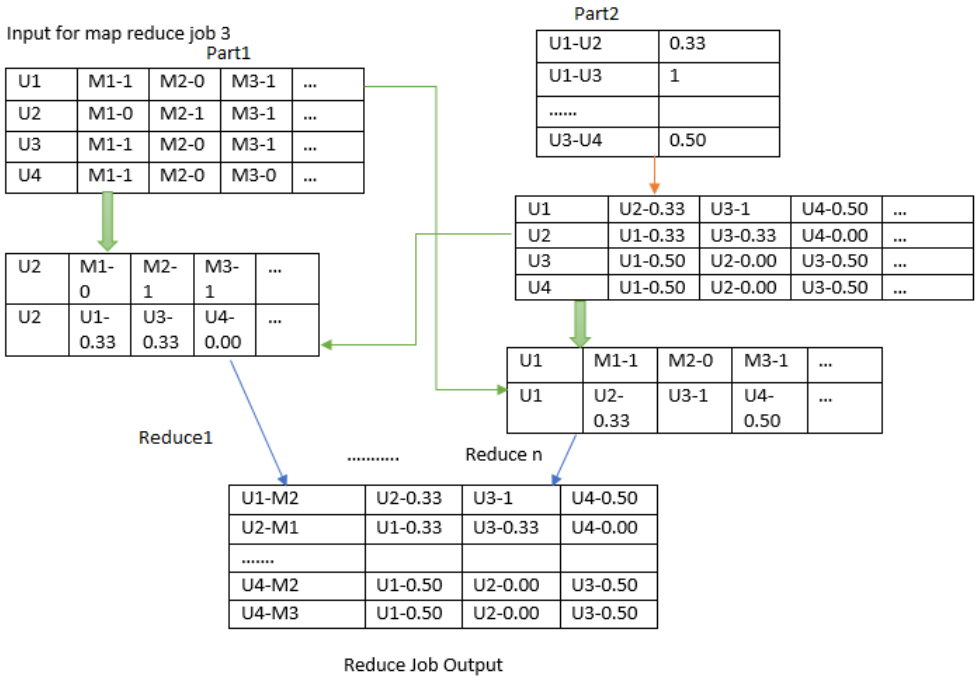

Figure 5.6: MRJob3- combine similarity of users to generate a common key user-item

The output of the reduce job will contain user-item as key for each user with all items and value as similarity values of all other users with the user present in key.

The figure 5.7 shows the final output for second map reduce job.

5.3.2 Map-reduce Job4

The next task it to build the user-item keys and values where the value contains the item-ratings of each user other than the user in key. Even in this map-reduce task we need to form the user-item keys from the existing item-user data for all the items that the user has not rated/purchased.

This job does not need a reduce task as there are no common keys generated by the mappers to be combined.

```
u1-m2:  u7~0.67;u6~0.75;u2~0.4;
u1-m2:  u10~0.33;u3~0.8;u9~0.4;
u1-m2:  u4~0.75;u8~0.43;u5~0.75;
u1-m4:  u7~0.67;u6~0.75;u2~0.4;
u1-m4:  u10~0.33;u3~0.8;u9~0.4;
u1-m4:  u4~0.75;u8~0.43;u5~0.75;
u1-m5:  u7~0.67;u6~0.75;u2~0.4;
u1-m5:  u10~0.33;u3~0.8;u9~0.4;
u1-m5:  u4~0.75;u8~0.43;u5~0.75;
u1-m7:  u7~0.67;u6~0.75;u2~0.4;
u1-m7:  u10~0.33;u3~0.8;u9~0.4;
u1-m7:  u4~0.75;u8~0.43;u5~0.75;
u1-m9:  u7~0.67;u6~0.75;u2~0.4;
u1-m9:  u10~0.33;u3~0.8;u9~0.4;
u1-m9:  u4~0.75;u8~0.43;u5~0.75;
u1-m10: u7~0.67;u6~0.75;u2~0.4;
u1-m10: u10~0.33;u3~0.8;u9~0.4;
u1-m10: u4~0.75;u8~0.43;u5~0.75;
u10-m1: u9~0.17;u1~0.33;u2~0.4;
u10-m1: u3~0.5;u4~0.4;u5~0.17;
u10-m1: u6~0.4;u7~0.43;u8~0.67;
u10-m2: u9~0.17;u1~0.33;u2~0.4;
u10-m2: u3~0.5;u4~0.4;u5~0.17;
u10-m2: u6~0.4;u7~0.43;u8~0.67;
u10-m3: u9~0.17;u1~0.33;u2~0.4;
u10-m3: u3~0.5;u4~0.4;u5~0.17;
```

Figure 5.7: Output of MR Job3 with sample data set

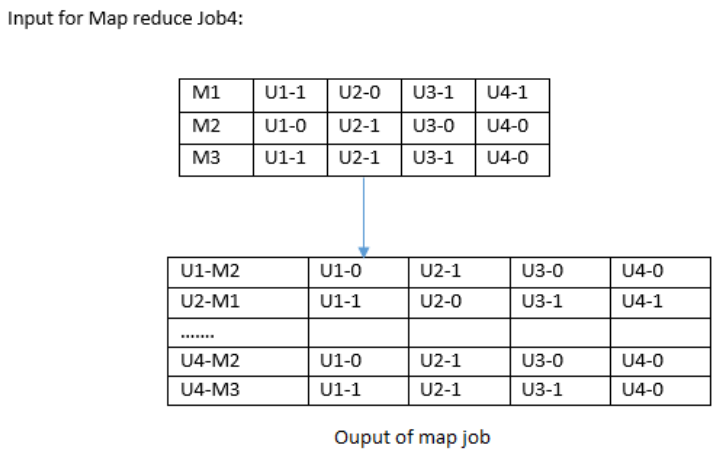

Figure 5.8: MRJob4 - generate a common key user-item with user ratings as value

This map jobs just generates new set of keys with same values. These new keys match with the keys generated by third map reduce job.

```
u1-m10: u1-0;u2-0;u3-0;u4-0;u5-0;u6-0;u7-0;u8-1;u9-1;u10-1
u1-m2:  u1-0;u2-0;u3-0;u4-0;u5-0;u6-0;u7-0;u8-1;u9-0;u10-0
u1-m4:  u1-0;u2-1;u3-1;u4-0;u5-0;u6-0;u7-1;u8-1;u9-0;u10-1
u1-m5:  u1-0;u2-0;u3-0;u4-0;u5-0;u6-0;u7-0;u8-0;u9-0;u10-0
u1-m7:  u1-0;u2-0;u3-0;u4-0;u5-0;u6-0;u7-0;u8-0;u9-0;u10-0
u1-m9:  u1-0;u2-0;u3-0;u4-0;u5-0;u6-0;u7-1;u8-0;u9-0;u10-0
u10-m1: u1-1;u2-0;u3-1;u4-1;u5-1;u6-0;u7-1;u8-0;u9-1;u10-0
u10-m2: u1-0;u2-0;u3-0;u4-0;u5-0;u6-0;u7-0;u8-1;u9-0;u10-0
u10-m3: u1-1;u2-1;u3-1;u4-0;u5-1;u6-1;u7-1;u8-1;u9-1;u10-0
u10-m5: u1-0;u2-0;u3-0;u4-0;u5-0;u6-0;u7-0;u8-0;u9-0;u10-0
u10-m7: u1-0;u2-0;u3-0;u4-0;u5-0;u6-0;u7-0;u8-0;u9-0;u10-0
u10-m9: u1-0;u2-0;u3-0;u4-0;u5-0;u6-0;u7-1;u8-0;u9-0;u10-0
u2-m10: u1-0;u2-0;u3-0;u4-0;u5-0;u6-0;u7-0;u8-1;u9-1;u10-1
u2-m1:  u1-1;u2-0;u3-1;u4-1;u5-1;u6-0;u7-1;u8-0;u9-1;u10-0
u2-m2:  u1-0;u2-0;u3-0;u4-0;u5-0;u6-0;u7-0;u8-1;u9-0;u10-0
u2-m5:  u1-0;u2-0;u3-0;u4-0;u5-0;u6-0;u7-0;u8-0;u9-0;u10-0
u2-m7:  u1-0;u2-0;u3-0;u4-0;u5-0;u6-0;u7-0;u8-0;u9-0;u10-0
u2-m8:  u1-1;u2-0;u3-1;u4-1;u5-1;u6-1;u7-1;u8-1;u9-0;u10-1
u2-m9:  u1-0;u2-0;u3-0;u4-0;u5-0;u6-0;u7-1;u8-0;u9-0;u10-0
u3-m10: u1-0;u2-0;u3-0;u4-0;u5-0;u6-0;u7-0;u8-1;u9-1;u10-1
u3-m2:  u1-0;u2-0;u3-0;u4-0;u5-0;u6-0;u7-0;u8-1;u9-0;u10-0
u3-m5:  u1-0;u2-0;u3-0;u4-0;u5-0;u6-0;u7-0;u8-0;u9-0;u10-0
u3-m7:  u1-0;u2-0;u3-0;u4-0;u5-0;u6-0;u7-0;u8-0;u9-0;u10-0
u3-m9:  u1-0;u2-0;u3-0;u4-0;u5-0;u6-0;u7-1;u8-0;u9-0;u10-0
u4-m10: u1-0;u2-0;u3-0;u4-0;u5-0;u6-0;u7-0;u8-1;u9-1;u10-1
```

Figure 5.9: Output of MR4 with sample data set

The figure 5.9 shows the final output for second map reduce job.

5.3.3 Map-reduce Job5

This job combines the results of previous two map-reduce jobs and combines the results to contain both item-raking and similarity of the all the other users with respect to user in key.

The map tasks processes the user-item keys and values from each file individually from two different locations and generates new user-item-user key and values. The value is the same previous value. The value can be either rating user-item or similarity of the user-pair.

The reduce task processes the keys from mapper. Each mapper has two same keys, one has the item rating and other has the user similarity. The mapper computes a product of these two values and saves the key as user-item and the product value as value.

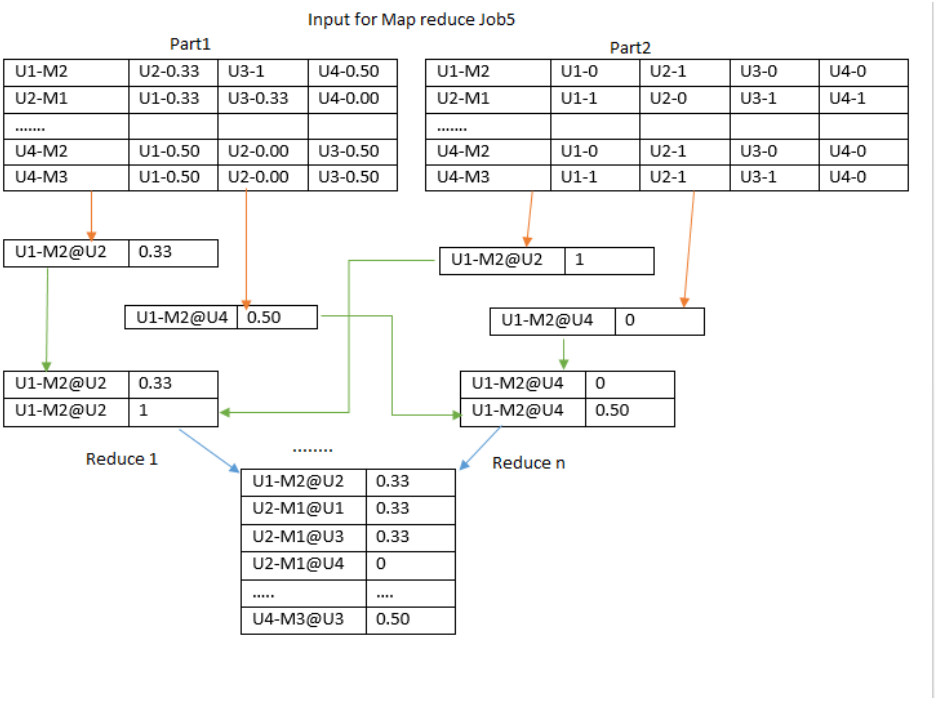

Figure 5.10: MRJob5- combine user ratings and similarities to compute ranking

This map reduce jobs computes the ranking of a particular item given by a particular user. The user with higher similarity value give higher rank to an item. The ranking computation here can happen in two ways based on the input. If the input for the map-reduce jobs is a user-item matrix which has binary values 0's representing non-rated item and '1' represents rating. Then the ranking given by a user is just the similarity index of the user with the active user(the user for whom the recommendation is being generated for). If the input for map reduce contains ratings instead of binary notation, then the similarity value and the rating of that item both of them will impact the ranking. A user with most similar taste will have highest ranked item as the item for which he has rated the highest rating.

The figure 5.11 shows the final output for second map reduce job.

```
u1-m10@ u10:0.33
u1-m10@ u1:0.0
u1-m10@ u2:0.0
u1-m10@ u3:0.0
u1-m10@ u4:0.0
u1-m10@ u5:0.0
u1-m10@ u6:0.0
u1-m10@ u7:0.0
u1-m10@ u8:0.43
u1-m10@ u9:0.4
u1-m2@  u10:0.0
u1-m2@  u1:0.0
u1-m2@  u2:0.0
u1-m2@  u3:0.0
u1-m2@  u4:0.0
u1-m2@  u5:0.0
u1-m2@  u6:0.0
u1-m2@  u7:0.0
u1-m2@  u8:0.43
u1-m2@  u9:0.0
u1-m4@  u10:0.33
u1-m4@  u1:0.0
u1-m4@  u2:0.4
u1-m4@  u3:0.8
u1-m4@  u4:0.0
```

Figure 5.11: Output of MR5 with sample data set

5.3.4 Map-reduce Job6

This task process the output generated by previous mapper and creates key such that the key contains only user and the value is combination of item and its ranking. So this mapper creates all the key,value pairs for the items that the user needs ranking for.

The reduce jobs combines all the values for same key and forms value by combining all the items and corresponding rankings for the user.

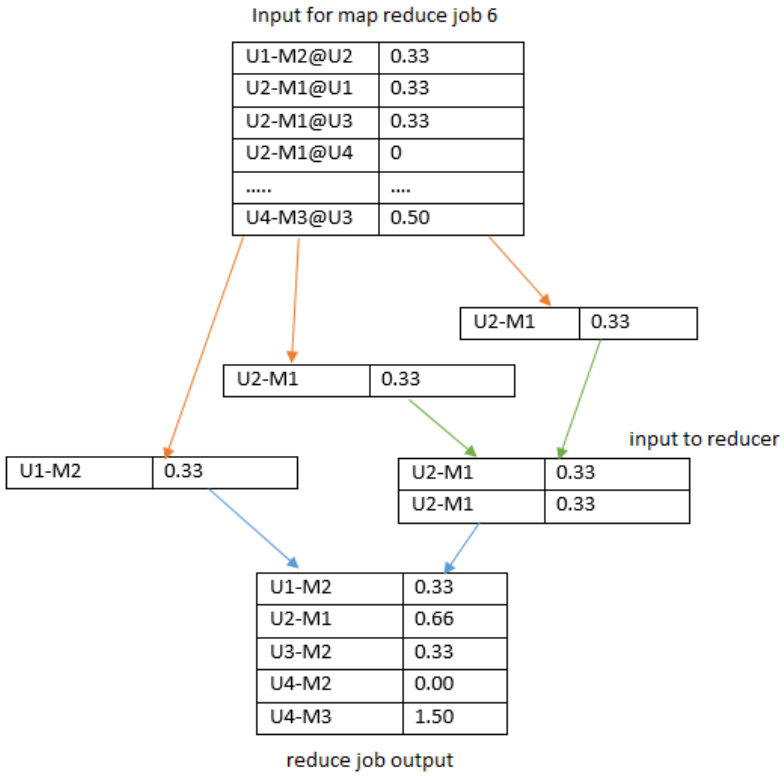

Figure 5.12: MRJob6- sum up the item rankings by item

The figure **??** shows the final output for second map reduce job.

5.3.5 Map-reduce Job7

The next map job creates a sort list of keys from the previous output. The output of the previous reduce job contains user as key and all items and their rankings. The current

```
u1-m10: 1.16
u1-m2:  0.43
u1-m4:  2.63
u1-m5:  0.00
u1-m7:  0.00
u1-m9:  0.67
u10-m1: 2.00
u10-m2: 0.67
u10-m3: 3.07
u10-m5: 0.00
u10-m7: 0.00
u10-m9: 0.43
u2-m10: 1.10
u2-m1:  2.10
u2-m2:  0.50
u2-m5:  0.00
u2-m7:  0.00
u2-m8:  3.30
u2-m9:  0.50
u3-m10: 1.40
u3-m2:  0.57
u3-m5:  0.00
u3-m7:  0.00
u3-m9:  0.83
u4-m10: 0.89
u4-m2:  0.29
```

Figure 5.13: Output of MR6 with sample data set

mapper processes each user record at a time by creating a sorted list of item-raking, so that the item with highest rank is the item the user is most likely interested in. Thus by end of all map-reduce tasks, the result contains all the users and item-raking for all the non-rated items.

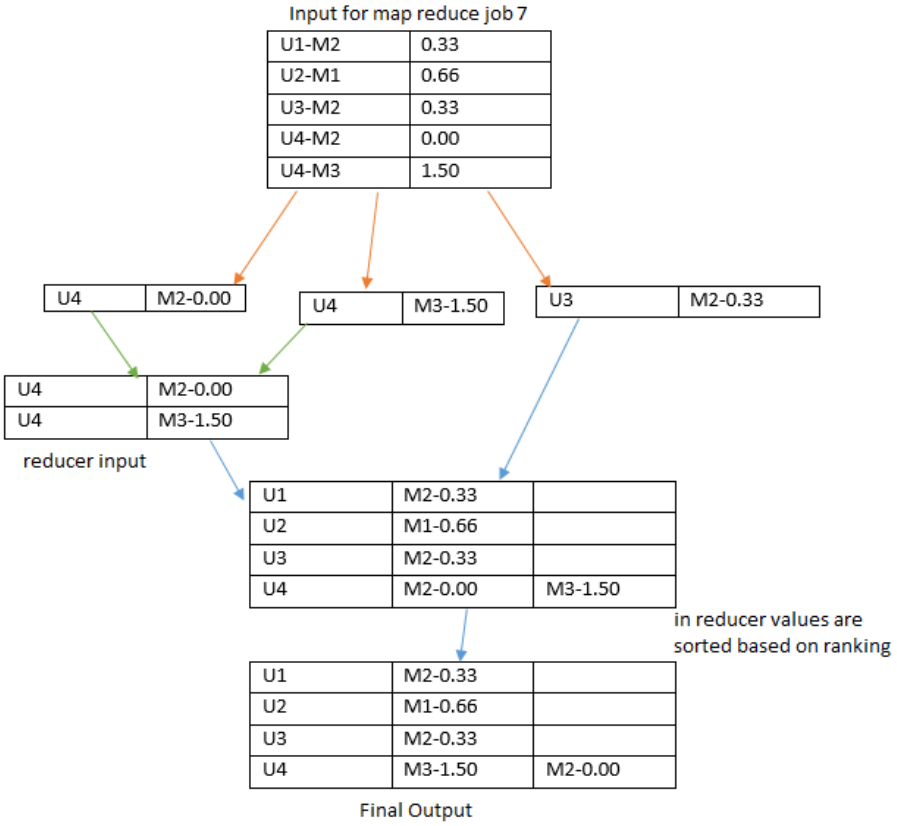

Figure 5.14: MRJob7- combine the rankings of same user and sort them

The figure **??** shows the final output for second map reduce job.

```
u10:    m3-3.07;m1-2.0;m2-0.67;m9-0.43;m5-0.0;m7-0.0;
u1:     m4-2.63;m10-1.16;m9-0.67;m2-0.43;m5-0.0;m7-0.0;
u2:     m8-3.3;m1-2.1;m10-1.1;m2-0.5;m9-0.5;m5-0.0;m7-0.0;
u3:     m10-1.4;m9-0.83;m2-0.57;m5-0.0;m7-0.0;
u4:     m3-3.54;m4-1.99;m10-0.89;m9-0.5;m2-0.29;m5-0.0;m7-0.0;
u5:     m6-3.51;m4-1.76;m10-0.96;m9-0.5;m2-0.29;m5-0.0;m7-0.0;
u6:     m1-3.05;m4-2.5;m10-1.1;m2-0.5;m9-0.5;m5-0.0;m7-0.0;
u7:     m10-1.22;m2-0.5;m5-0.0;m7-0.0;
u8:     m1-2.37;m9-0.5;m5-0.0;m7-0.0;
u9:     m8-2.38;m6-2.08;m4-1.28;m2-0.29;m9-0.29;m5-0.0;m7-0.0;
```

Figure 5.15: Output of MR7 with sample data set

Chapter 6

Experiments

The algorithm is designed to derive recommendations to the users based on their similarity with the other users. To derive recommendations two different types of input data is used for testing. The experiments will use both the binary notation data and also the rating data to derive the recommendations. The similarity computation does not change as the computation counts a non-zero rating as 1.

Experiments are done on both single node and multi-node clusters with varying data sizes and performance metrics are recorded. All the map-reduce jobs are run in sequence as the output of the previous jobs are given as input for the next job.

6.1 Single Node Cluster

Single Node Cluster has one of each HDFS components: JobTracker, TaskTracker, NameNode, Secondary NameNode, DataNode.

A series of steps need to be executed to completely test the algorithm. We can write a shell script to execute them in a row.

Command to copy the test data from local system to Hadoop File System.

```
bin/hadoop fs -copyFromLocal /local-file-path /user/sudha/
```

Figure 6.1: Command to copy data to HDFS

Sample command to run a job on Hadoop cluster:

```
bin/hadoop jar UBRecSys.jar com/rec/userbased/RecGenerator
/user/sudha/recsys/merged/item/output /user/sudha/recsys/merged/user/output
/user/sudha/recsys/merged/output
```

Figure 6.2: Command to run hadoop job on hadoop cluster

On a single node-cluster, experiments are conducted using different input data sets. The results of experiments on the single node cluster can be considered as sequential experiment results as the same processor is handling all the map reduce jobs.

6.1.1 Test Data:Item-wise partition

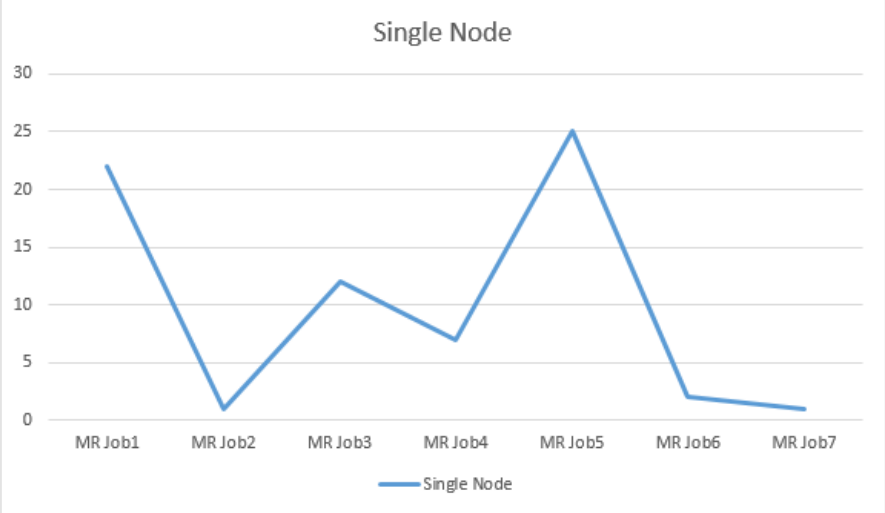

Figure 6.3: Single Node Cluster Test Data1

The chart 6.3 show the processing time of each map-reduce job on a single node cluster.

6.1.2 Test Data:Item-wise and User-wise partition

The second round of experiments are conducted using the test data with is partitioned using both item-wise and user-wise strategies. This improves the performance of map-reduce jobs in the second part of the algorithm. Especially we can see huge difference in the processing time of MapReduceJob5.

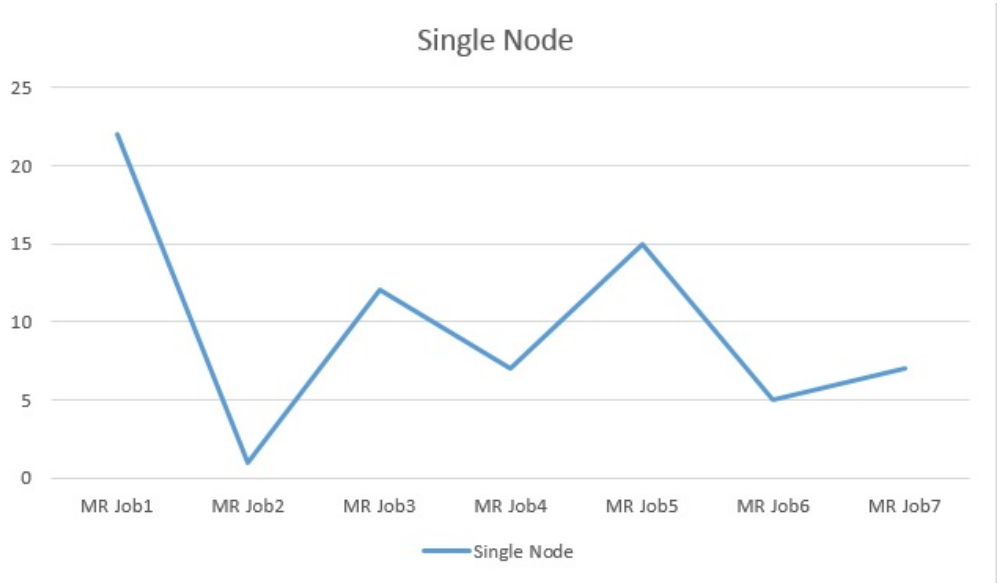

Figure 6.4: Single Node Cluster Test Data 2

The figure 6.4 shows the processing time taken by each map-reduce job.

6.2 Multi Node Cluster

A multi-node cluster is a collection of single node clusters, in which one of the nodes acts as a master and rest of them are slaves.

There exists a DataNode and TaskTracker on each slave node and a master node has NameNode and JobTracker. Master node can also have other components DataNode and TaskTracker in case it needs to handle some processing itself. A master can also act as Secondary NameNode or we can have a different node to serve as secondary NameNode.

The following experiments were conducted on a Hadoop cluster with a MapReduce implementation of our approach.

Experiments with different datasets are conducted on clusters with 2 nodes and 3 nodes.

6.2.1 2Node Cluster

A multi-node cluster is set up to test the algorithm performance with large data sets. The cluster has one master and 1 slave nodes. The master node triggers the job and assign tasks to the slaves. The master is runs the JobTracker and Namenode and Secondary Namenode. Slaves and master all server as Datanodes,TaskTrackers.

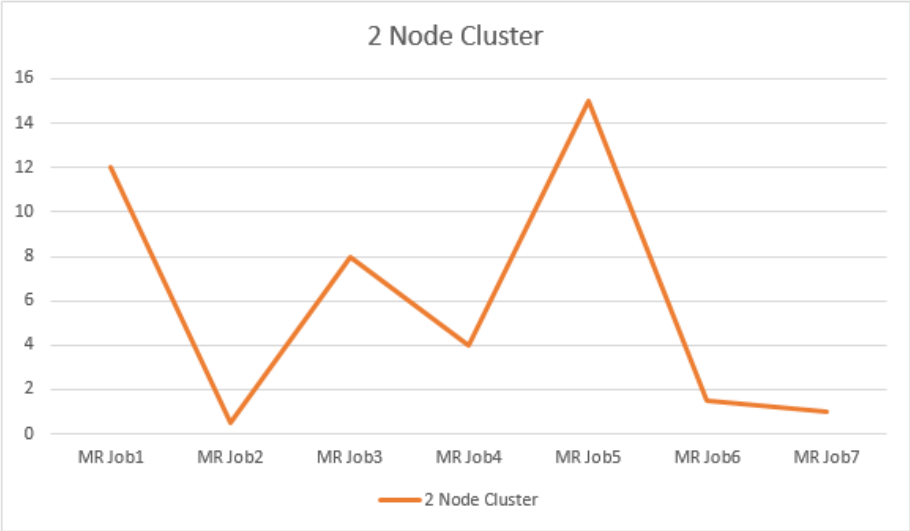

Figure 6.5: 2 Node Cluster Map reduce job processing times

The chart 6.5 show the processing times of the Map reduce jobs on 2 node clusters.

6.2.2 3Node Cluster

A multi-node cluster is set up to test the algorithm performance with large data sets. The cluster has one master and 2 slave nodes.

The chart 6.6 show the processing times of the Map reduce jobs on 3 node clusters.

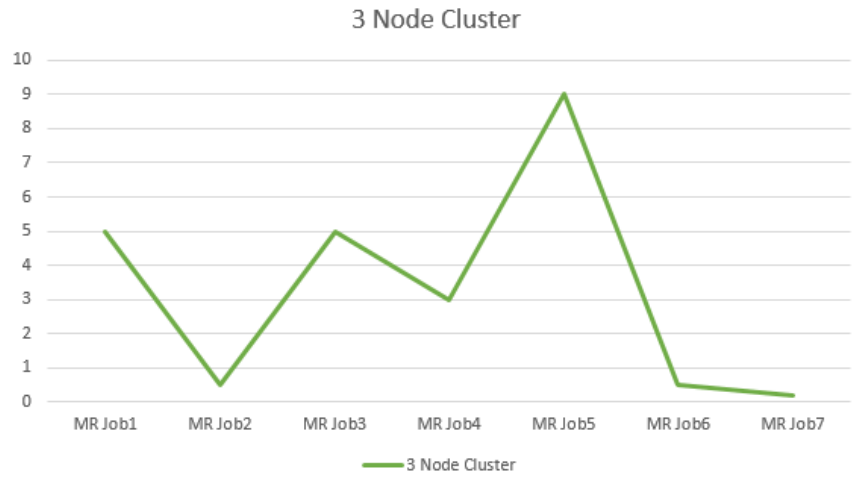

Figure 6.6: 3 Node Cluster Map reduce job processing times

6.2.3 Comparison of results

Comparison of results of two different data sets on all the three cluster setups.

Two different data sets of different sizes (number of users and number of items) are tested on the cluster setup. The ratings count is about 1M and 1.5M. The data sets are ranging from 0.8 GB to 3GB in size. The processing time for all the map reduce jobs is recorded and shown below.

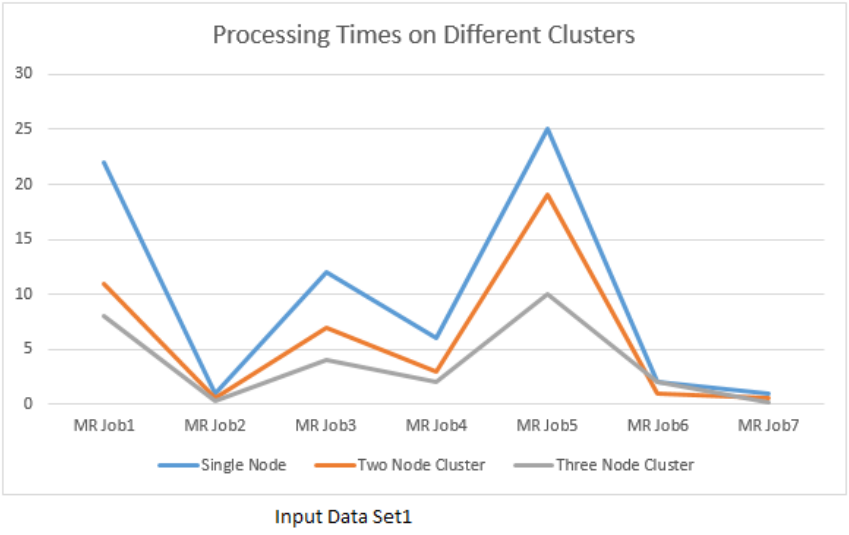

Figure 6.7: Map reduce job processing times for input data set 2

The chart shown in figure 6.7 shows the results for input data set for 1M ratings. The processing times for all map reduce jobs are recorded.

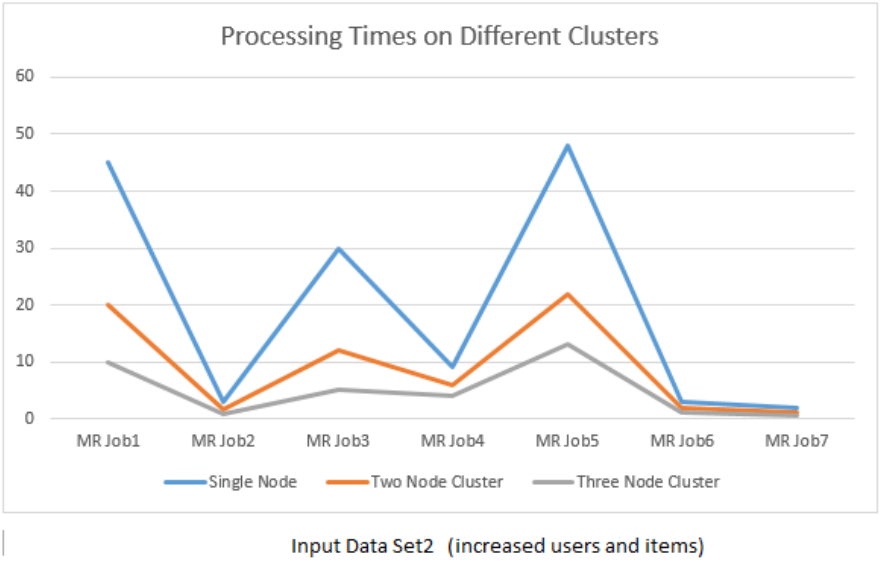

Figure 6.8: Map reduce job processing times for input data set 2

The chart shown in 6.8 shows the results for input data set for 1.5M ratings. The processing times for all map reduce jobs are recorded.

6.3 Limitations

The input data must be preprocessed in the required format. The experiments does not include the effort for preprocessing as the performance measurement metric. Only the running time and data size are considered.

Chapter 7

Future Work

The current implementation of the algorithms uses Jaccard similarity measure. There can be other similarity computations which can be parallelized to identify similar users. As the recommendation is mainly based on similarity any improvements in similarity computation might improve the algorithm accuracy.

The algorithm uses only rating of the item given by the user to compute the similarity, but there can be other parameters which are worth considering to improve the accuracy of the recommendations. But we need to identify a way to compute similarity using a multi-dimensional vector in parallel using Hadoop.

Chapter 8

Conclusion

In this thesis we discussed about recommendation system algorithms and map reduce programming model by hadoop. A sequential algorithm to compute user-based similarity is explained. We also discussed how to convert the sequential algorithm into a parallel algorithm using hadoop map-reduce framework. The idea to parallelize the computation is inspired by wordcount example of mapreduce. The computation is split into multiple smallwe tasks and each of the small task is handled by a mep-reduce job. In the thesis we showed details of the approach including the inputs and outputs produced at each phase of execution.

Different experiments are conducted to show the performance improvement of using a multi-node cluster. Test data for experiments is taken from group lens and movie lens data bases. Experiments are done using both the partition strategies. The results show that when the second partition strategy (user-based) is also implemented, the performance is far better than with the single partition based on items. The Hadoop map-reduce approach saves a lot of resources in computing similarities and generates recommendations in short time.

Bibliography

[1] Konstan, J., Miller, B., Maltz, D., Herlocker, J., Gordon, L., and Riedl, J. (2012). GroupLens: Applying Collaborative Filtering to Usenet News. Communications of the ACM, 40(3), pp. 77-87.. [Online; accessed June 2014].

[2] Resnick, P., Iacovou, N., Suchak, M., Bergstrom, P., and Riedl, J. (1994). GroupLens: An Open Architecture for Collaborative Filtering of Netnews.. [Online; accessed August 2014].

[3] Shardanand, U., and Maes, P. (1995). Social Information Filtering: Algorithms for Automating Word of Mouth.. [Online; accessed September 2014].

[4] GroupLens Research Group. Recommender Systems for Large-scale E-Commerce: Scalable Neighborhood Formation Using Clustering.. [Online; accessed October 2014].

[5] A. Metwally and C. Faloutsos. (2012). V-smart-join: A scalable MapReduce framework for all-pair similarity joins of multisets and vectors.. [Online; accessed October 2014].

[6] Jeffrey Dean and Sanjay Ghemawat, Google, Inc. MapReduce: Simplified Data Processing on Large Clusters.. [Online; accessed November 2014].

[7] J. Herlocker, J. Konstan, L. Terveen, and J. Riedl. Evaluating collaborative filtering recommender systems. ACM Transactions on Information Systems.. [Online; accessed November 2014].

[8] Adomavicius G., Tuzhilin A. (2005). Toward the next generation of recommender systems: A survey of the state-of-the-art and possible extensions.. [Online; accessed December 2014].

[9] J. Dean and S. Ghemawat. (2008). Mapreduce: Simplified data processing on large clusters.. [Online; accessed December 2014].

[10] Apache Software Foundation. Hadoop.. `http://hadoop.apache.org/hadoop`. [Online; accessed May 2014].

[11] Douglas Thain, Todd Tannenbaum, and Miron Livny. (2005). Distributed computing in practice: the condor experience: Research articles. [Online; accessed December 2014].

[12] Apache Hadoop. Hadoop Distributed File Systems (HDFS). `http://hadoop.apache.org/docs/r0.17.1/hdfs_design.html`, (1997). [Online; accessed December 2014].

[13] Brian F. Cooper, Raghu Ramakrishnan, Utkarsh Srivastava, Adam Silberstein, Philip Bohannon, Hans arno Jacobsen, Nick Puz, Daniel Weaver, and Ramana Yerneni. (2008). Pnuts: Yahoo!s hosted data serving platform.. [Online; accessed May 2014].

[14] Dean and S. Ghemawat. (2008). Mapreduce: Simplified data processing on large clusters.. [Online; accessed January 2014].

[15] Yahoo. Yahoo! launches worldis largest hadoop production application.. `http://tinyurl.com/2hgzv7`. [Online; accessed January 2014].

[16] Apache Software Foundation. The hive project.. `http://hadoop.apache.org/hive`. [Online; accessed July 2014].

[17] Apache Software Foundation. The pig project.. `http://hadoop.apache.org/pig`. [Online; accessed July 2014].

[18] Apache Software Foundation. Apache Hadoop.. `http://hadoop.apache.org/zookeeper`. [Online; accessed July 2014].

[19] Shang Ming-Sheng, Zhang Zi-ke. (2009). Diffusion-Based Recommendation in Collaborative Tagging Systems.Chin.. [Online; accessed June 2014].

[20] Shang Ming-Sheng, Jin Ci-Hang, Zhou Tao, Zhang Yi- Cheng. (2009). Collaborative filtering based on multi-channel diffusion. Physics A: Statistical Mechanics and its Applications.. [Online; accessed July 2014].

[21] Isard M, Budiu M, Yu Y, Birrell A, Fetterly D. Dryad. (2007). Distributed data-parallel programs from sequential building blocks.. [Online; accessed May 2014].

[22] DeCandia G, Hastorun D, Jampani M, Kakulapati G, Lakshman A, Pilchin A, Sivasubramanian S, Vosshall P, Vogels W. (2007). Dynamo: Amazons highly available key-value store.. [Online; accessed February 2014].

[23] Chu LK, Tang H, Yang T, Shen K. (2003). Optimizing data aggregation for cluster-based Internet services.. [Online; accessed February 2014].

[24] Dean J, Ghemawat S. (2007). Distributed programming with Mapreduce.. [Online; accessed February 2014].

[25] Dean J, Ghemawat S. (2007). MapReduce: Simplified data processing on large clusters.. [Online; accessed January 2014].

[26] Ghemawat S, Gobioff H, Leung ST. The Google file system.. [Online; accessed January 2014].

[27] Brad Hedlund. Understanding Hadoop Clusters and the Network. `www-01.ibm.com/software/data/infosphere/hadoop/`. [Online; created December 2014].

[28] IBM. IBM Hadoop. `http://www.ibm.com/cloud-computing/us/en/what-is-cloud-computing.html`, 2012. [Online; accessed December 2014].

[29] Yahoo! Hadoop Introduction. http://developer.yahoo.com/hadoop/tutorial/module1.html, 2012. [Online; accessed December 2014].

[30] Yahoo! Hadoop MapReduce. http://developer.yahoo.com/hadoop/tutorial/module4.html#dataflow, 2012. [Online; accessed December 2014].

[31] Yahoo! Hadoop MapReduce Basics. http://developer.yahoo.com/hadoop/tutorial/module4.html#basics, 2012. [Online; accessed December 2014].